



1

## Liberty Bindings and Profiles Specification

2

Version 1.1 – 16

3

18 November 2002

4

5 **Document Description:** draft-liberty-architecture-bindings-and-profiles-1.1-16.doc.

6 **Notice**

7 Copyright © 2002 ActivCard; American Express Travel Related Services; America Online, Inc.;  
8 Bank of America; Bell Canada; Cingular Wireless; Cisco Systems, Inc.; Citigroup; Communicator,  
9 Inc.; Consignia; Deloitte & Touche LLP; EarthLink, Inc.; Electronic Data Systems, Inc.; Entrust,  
10 Inc.; Ericsson; Fidelity Investments; France Telecom; Gemplus; General Motors; Hewlett-Packard  
11 Company; i2 Technologies, Inc.; Intuit Inc.; MasterCard International; NEC Corporation; Netegrity;  
12 NeuStar; Nextel Communications; Nippon Telegraph and Telephone Company; Nokia Corporation;  
13 Novell, Inc.; NTT DoCoMo, Inc.; OneName Corporation; Openwave Systems Inc.;  
14 PricewaterhouseCoopers LLP; Register.com; RSA Security Inc; Sabre Holdings Corporation; SAP  
15 AG; SchlumbergerSema; SK Telecom; Sony Corporation; Sun Microsystems, Inc.; United Airlines;  
16 VeriSign, Inc.; Visa International; Vodafone Group Plc; Wave Systems. All rights reserved.

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

22 Implementation of the Specifications may involve the use of one or more of the following United  
23 States Patents claimed by AOL Time Warner, Inc.: No.5,774,670, No.6,134,592, No.5,826,242, No.  
24 5,825,890, and No.5,671,279. The Sponsors of the Specification take no position concerning the  
25 evidence, validity or scope of the claimed subject matter of the aforementioned patents.

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

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

35 Liberty Alliance Project  
36 Licensing Administrator  
37 c/o IEEE-ISTO  
38 445 Hoes Lane, P.O. Box 1331  
39 Piscataway, NJ 08855-1331, USA

40

40  
41  
42

42 **Editors**

43           Jason Rouault, Hewlett-Packard Company  
44           Tom Wason. IEEE-ISTO

45 **Contributors**

46  
47

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

48

48 Document History

Version #	Date	Editor	Scope of changes
1.0	14-Mar-02	Jason Rouault	Initial Draft Based on Liberty V1.0
1.1	04-Nov-02	Jason Rouault	<p>CR1097: Typo corrections.</p> <p>CR1098: collapse the HTTP-GET-Based and HTTP-Redirect-Based profiles into the same profile</p> <p>CR1106: prescriptive alternative for countermeasure to Unsigned &lt;lib:AuthnRequest&gt; message. Section 4.4.2.1.</p> <p>CR1118: Added rogue participant threat notice. Section 4.4.2.3.</p> <p>CR1122: Describe rogue impersonation threat and countermeasure. Section 4.4.2.1.</p> <p>CR1130: Resolution of LEC vulnerability to a man-in-the-middle attack. Section 3.2.5.1.</p> <p>CR1138: Changed namespace notation "http://projectliberty.org/schemas/core/2002/08 to "http://projectliberty.org/schemas/core/2002/12. Section 1.1, Section 3.2.5.2 step 3.</p> <p>CR1139: Change cookie constraints. In Section 3.6.1 changed "identity provider ID" to "identity provider succinct ID". Restricted Path prefix. In Section 3.6.2 removed figure and steps, added text describing setting common domain cookie. In section 3.6.3 removed steps and figure, inserted text describing reading the common domain cookie.</p> <p>CR1140: Clarification of construction of ProviderSuccinctID. Section 3.2.2.2</p> <p>CR1147: Correct typo.</p> <p>CR1148: Remove use of element samlp:RespondWith. Section 3.1.2.1.2.</p> <p>CR1150: Added note on escaping base64-encoded signature values. Section 3.1.2.1.</p> <p>CR1151: Added "The service URL provided by the provider (the URL to which &lt;query&gt; parameters are added) MUST NOT contain any pre-existing &lt;query&gt; parameter values." Section 3.1.2.1</p> <p>CR1156: URI specified in the service provider</p>

			<p>metadata element FederationTerminationNotificationProtocolProfile. Section 3.4.1.</p> <p>CR1157: Session persistence at federation termination notification. Section 3.4.2</p> <p>CR1159: Changed "...SAML artifact on success or empty on failure." To "...SAML artifact on success or on failure." Added following sentence: "In the case of failure, the status information will be communicated in the &lt;samlp:Response&gt; returned." in step 9. Section 3.2.2.1</p> <p>CR1160, CR1161: Generalizing Register Name Identifier Profile. Register name identifier profile is changed to support NameIdentifier changes by both Identity Provider and Service Provider. Section 3.3. Change Name Registration Profile description to reflect the bi-directional nature of the profile for establishing or changing a Principal's name identifier. Section 3.</p> <p>CR1162: Change "Name" to "NameIdentifier". Sections 3.1.2.1.3, 3.1.2.1.4</p> <p>CR1166: Changed example to provide correct content type. Section 2.2.</p> <p>CR1167: Change header from "Liberty-LECP" to "Liberty-Enabled". Section 3.2.5.1.</p> <p>CR1168: Delete requirement for base64 encoding in Section 3.2.5.2, Step 3.</p> <p>CR1169: Change "...transfer service responds and <b>redirects</b>..." to "...transfer service responds and <b>sends</b>...", delete redirection rules from Step 3. Added, "The form and contents of the HTTP response in this step are profile-dependent." in following line. Removed "attached to the URL fulfilling the redirect request" from step 4. Section 3.2.1. Added step 3 to Sections 3.2.2.1 and 3.2.3.</p> <p>CR1178: Changed "Maximum URL length of 256 bytes." To "Minimum maximum URL length of 256 bytes." Section 3.1.1.</p> <p>CR1180: Modified Step 5: Processing &lt;AuthnRequest&gt;, Section 3.2.1.</p> <p>CR1181: Modified Step 6: HTTP Response with &lt;AuthnResponse&gt; or Artifact, Section 3.2.1.</p> <p>CR1182: Changed common requirements 3, 4 &amp; 6, deleted #12. Section 3.1.</p>
--	--	--	--

			<p>CR1219: Changed references to Name Registration Profiles, now plural. Sections 3, 3.3.</p> <p>CR1220: Specify algorithms and identifiers, Section 3.1.2.1</p> <p>CR1223: Correct typos. Sections 3.2.1, 3.3, 3.5, 3.5.2.2.</p> <p>CR1224: Replaced <code>&lt;lib:Assertion&gt;</code> with <code>&lt;saml:Assertion&gt;</code> 8 places.</p> <p>CR1228: Removed “asynchronous”, Section 3.3.2.2.</p> <p>CR1229: Revised code for the original <code>&lt;lib:AuthnRequest&gt;</code> message, Section 3.1.2.1.2.</p> <p>CR1230: Changed Liberty-LECP to Liberty-Enabled, 2 places in Section 3.2.5.1.</p> <p>CR1231: Added URI-based identifiers to Register Name Identifier profiles, Sections 3.3.2.1, 3.3.2.2</p> <p>CR1232: Typo</p> <p>CR1233: Added 2 metadata elements to Register Name Identifier profiles, Section 3.3</p>
--	--	--	---

49

50

50 **Table of Contents**

51	1	Introduction .....	9
52	1.1	Notation .....	9
53	2	Protocol Bindings .....	9
54	2.1	SOAP Binding for Liberty .....	10
55	2.2	Example of Message Exchange Using SOAP over HTTP .....	10
56	3	Profiles .....	12
57	3.1	Common Requirements .....	12
58	3.1.1	User Agent .....	13
59	3.1.2	Formatting and Encoding of Protocol Messages .....	13
60	3.1.3	Provider Metadata .....	16
61	3.2	Single Sign-On and Federation Profiles .....	16
62	3.2.1	Common Interactions and Processing Rules .....	17
63	3.2.2	Liberty Browser Artifact Profile .....	20
64	3.2.3	Liberty Browser POST Profile .....	24
65	3.2.4	Liberty WML POST Profile .....	26
66	3.2.5	Liberty-Enabled Client and Proxy Profile .....	29
67	3.3	Register Name Identifier Profile .....	34
68	3.3.1	Register Name Identifier Initiated at Identity Provider .....	35
69	3.3.2	Register Name Identifier Initiated at Service Provider .....	39
70	3.4	Identity Federation Termination Notification Profiles .....	40
71	3.4.1	Federation Termination Notification Initiated at Identity Provider .....	41
72	3.4.2	Federation Termination Notification Initiated at Service Provider .....	45
73	3.5	Single Logout Profiles .....	46
74	3.5.1	Single Logout Initiated at Identity Provider .....	47
75	3.5.2	Single Logout Initiated at Service Provider .....	53
76	3.6	Identity Provider Introduction .....	57
77	3.6.1	Common Domain Cookie .....	58
78	3.6.2	Setting the Common Domain Cookie .....	58
79	3.6.3	Obtaining the Common Domain Cookie .....	58
80	4	Security Considerations .....	59
81	4.1	Introduction .....	59
82	4.2	General Requirements .....	59
83	4.2.1	Security of SSL and TLS .....	59
84	4.2.2	Security Implementation .....	60
85	4.3	Classification of Threats .....	60
86	4.3.1	Threat Model .....	60
87	4.3.2	Rogue and Spurious Entities .....	60
88	4.3.3	Active and Passive Attackers .....	60
89	4.3.4	Scenarios .....	61
90	4.4	Threat Scenarios and Countermeasures .....	61
91	4.4.1	Common Threats for All Profiles .....	61
92	4.4.2	Single Sign-On and Federation .....	62
93	4.4.3	Name Registration .....	65
94	4.4.4	Federation Termination: HTTP-Redirect-Based Profile .....	65
95	4.4.5	Single Logout: HTTP-Redirect-Based Profile .....	65
96	4.4.6	Identity Provider Introduction .....	66
97	5	References .....	66



## 99 1 Introduction

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

### 105 1.1 Notation

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

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

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

116

117 Example code listings appear like this.

118

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

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

- 123 • The prefix `lib:` stands for the Liberty namespace  
124 `http://projectliberty.org/schemas/core/2002/12`
- 125 • The prefix `saml:` stands for the SAML assertion namespace (see [\[SAMLCore\]](#)).
- 126 • The prefix `samlp:` stands for the SAML request-response protocol namespace (see  
127 [\[SAMLCore\]](#)).
- 128 • The prefix `ds:` stands for the W3C XML signature namespace,  
129 `http://www.w3.org/2000/09/xmldsig#` (see [\[XMLSig\]](#)).
- 130 • The prefix `SOAP-ENV:` stands for the SOAP 1.1 namespace,  
131 `http://schemas.xmlsoap.org/soap/envelope` (see [\[SOAP1.1\]](#)).

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

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

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

## 137 2 Protocol Bindings

138 The Liberty protocol bindings are defined in this section.

## 139 2.1 SOAP Binding for Liberty

140 Because the Liberty protocols are an extension of the SAML protocol (see [[SAMLCore](#)]) and a  
141 SOAP protocol binding for SAML has been defined, the SOAP binding for Liberty MUST adhere to  
142 the processing rules for the “SOAP binding for SAML” as specified in [[SAMLBind](#)] unless  
143 otherwise noted. Just like SAML, the SOAP binding for Liberty uses HTTP as the transport  
144 mechanism.

## 145 2.2 Example of Message Exchange Using SOAP over HTTP

146 The following is an example of the SOAP exchange for the single sign-on browser artifact profile  
147 requesting an authentication assertion.

```
148 POST /authn HTTP/1.1
149 Host: idp.example.com
150 Content-type: text/xml
151 Content-length: nnnn
152 <soap-env:Envelope
153   xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
154   <soap-env:Header/>
155   <soap-env:Body>
156     <samlp:Request
157       xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
158       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
159       IssueInstant="2002-10-31T21:42:14Z" MajorVersion="1" MinorVersion="0"
160       RequestID="2H+PRhYSFYXozOD6r6PZ4YqyKfft">
161       <samlp:RespondWith
162         xmlns:lib="http://projectliberty.org/schemas/core/2002/12">
163         lib:AuthenticationStatementType
164       </samlp:RespondWith>
165       <ds:Signature>
166         <ds:SignedInfo>
167           <ds:CanonicalizationMethod
168             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
169           </ds:CanonicalizationMethod>
170           <ds:SignatureMethod
171             Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
172           </ds:SignatureMethod>
173           <ds:Reference URI="">
174             <ds:Transforms>
175               <ds:Transform
176                 Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
177               </ds:Transform>
178               <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
179               </ds:Transform>
180             </ds:Transforms>
181             <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
182             </ds:DigestMethod>
183             <ds:DigestValue>PdWEA3Zo+o00vDTpKc7IA5IAR3I=</ds:DigestValue>
184           </ds:Reference>
185         </ds:SignedInfo>
186         <ds:SignatureValue>
187           fODJy9kpGAlPF7T/T6EO/
188           zqe2DGYOXI29GUTPqSAwYlX1bGucVL0wuJ3rJbM8yabjmR9qIcUgX6
189           oslj9bmQAN/BS4aCcPaT+ud/OiptQYGSWKy3vrY2vocYRt4FeM/
190           jzJf7lxGRuDoaZ8K2laeOB9Qw
191           9JARxuVTBAjnuY8Zmd0=
192         </ds:SignatureValue>
193       </ds:Signature>
194       <samlp:AssertionArtifact>
195       AAOxpKP/m3nFdRE+CcOtXvH9ttzk8mViuGWIAPowqu/2jv1aIr/heKfX
196       </samlp:AssertionArtifact>
197     </samlp:Request>
198   </soap-env:Body>
199 </soap-env:Envelope>
```

200

201  
202 The following is an example of the corresponding response, which supplies an assertion containing  
203 the authentication statement as requested.

```
204 HTTP/1.1 200 OK
205 Content-Type: text/xml
206 Content-Length: nnnn
207 <soap-env:Envelope
208   xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
209   <soap-env:Header/>
210   <soap-env:Body>
211     <sampl:Response
212       xmlns:sampl="urn:oasis:names:tc:SAML:1.0:protocol"
213       InResponseTo="RPCUk211+GVz+t11LURp51oFvJXk"
214       IssueInstant="2002-10-31T21:42:13Z" MajorVersion="1" MinorVersion="0"
215       Recipient="http://localhost:8080/sp"
216       ResponseID="LANWfL2xLybnc+BCwgY+pl/vIVAj">
217     <sampl:Status>
218       <sampl:StatusCode
219         xmlns:qns="urn:oasis:names:tc:SAML:1.0:protocol"
220         Value="qns:Success">
221       </sampl:StatusCode>
222     </sampl:Status>
223     <saml:Assertion
224       xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
225       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
226       xmlns:lib="http://projectliberty.org/schemas/core/2002/12"
227       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
228       AssertionID="SqMC8Hs2vJ7Z+t4UiLSmhKOSU00U"
229       InResponseTo="RPCUk211+GVz+t11LURp51oFvJXk"
230       IssueInstant="2002-10-31T21:42:13Z" Issuer="http://localhost:8080/idp"
231       MajorVersion="1" MinorVersion="0"
232       xsi:type="lib:AssertionType">
233     <saml:Conditions
234       NotBefore="2002-10-31T21:42:12Z"
235       NotOnOrAfter="2002-10-31T21:42:43Z">
236     <saml:AudienceRestrictionCondition>
237       <saml:Audience>http://localhost:8080/sp</saml:Audience>
238     </saml:AudienceRestrictionCondition>
239     </saml:Conditions>
240     <saml:AuthenticationStatement
241       AuthenticationInstant="2002-10-31T21:42:13Z"
242       AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
243       xsi:type="lib:AuthenticationStatementType">
244     <saml:Subject xsi:type="lib:SubjectType">
245       <saml:NameIdentifier>C9FfGouQdBJ7bpkismYgd8ygeVb3PlWK</saml:NameIdentifier>
246       <saml:SubjectConfirmation>
247         <saml:ConfirmationMethod
248           urn:oasis:names:tc:SAML:1.0:cm:artifact-01
249         </saml:ConfirmationMethod>
250       </saml:SubjectConfirmation>
251       <lib:IDPProvidedNameIdentifier>
252         C9FfGouQdBJ7bpkismYgd8ygeVb3PlWK
253       </lib:IDPProvidedNameIdentifier>
254     </saml:Subject>
255     </saml:AuthenticationStatement>
256     <ds:Signature>
257       <ds:SignedInfo>
258         <ds:CanonicalizationMethod
259           Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
260         </ds:CanonicalizationMethod>
261         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
262         </ds:SignatureMethod>
263         <ds:Reference URI="">
264           <ds:Transforms>
265             <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
266             </ds:Transform>
267             <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
268             </ds:Transform>
269           </ds:Transforms>
270           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
271           </ds:DigestMethod>
272           <ds:DigestValue>ZbscbqHTX9H8bBftRIWlG4EpvlA=</ds:DigestValue>
273         </ds:Reference>
274       </ds:SignedInfo>
275       <ds:SignatureValue>
276         H+q3nC3jUaljl1uKUVkcC4iTFClxeZQIFf0nvHgPS5oZhtkBaDb9qITA7gIkotaB584wXTXwsfsu
277         IrwT5uL3r85Rj7IF6NeCeiy3K0+z3uewxyeZPz8wna449VNm0qNHYkgNak9ViNcp0/ks5MAttoPo
278         2iLOfaKu3wWG6d1G+DM=
279       </ds:SignatureValue>
280     </ds:Signature>
```

```
281     </saml:Assertion>  
282   </samlp:Response>  
283 </soap-env:Body>  
284 </soap-env:Envelope>  
285  
286
```

## 287 3 Profiles

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

292 The following profile categories are defined in this document:

- 293 • **Single Sign-On and Federation:** The profiles by which a service provider obtains an  
294 authentication assertion from an identity provider facilitating single sign-on and identity  
295 federation.
- 296 • **Name Registration:** The profiles by which service providers and identity providers specify  
297 the name identifier to be used when communicating with each other about the Principal.
- 298 • **Identity Termination Notification:** The profiles by which service providers and identity  
299 providers are notified of federation termination.
- 300 • **Single Logout:** The profiles by which service providers and identity providers are notified of  
301 authenticated session termination.
- 302 • **Identity Provider Introduction:** The profile by which a service provider discovers which  
303 identity providers a Principal may be using.

### 304 3.1 Common Requirements

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

- 307 1 All HTTP requests and responses MUST be drawn from either HTTP 1.1 (see [[RFC2616](#)]) or  
308 HTTP 1.0 (see [[RFC1945](#)]). When an HTTP redirect is specified, the HTTP response MUST  
309 have a status code of “302.” According to HTTP 1.1 and HTTP 1.0, the use of status code 302 is  
310 recommended to indicate “the requested resource resides temporarily under a different URI.”  
311 The response may also include additional headers and an optional message.
- 312 2 When `https` is specified as the `<scheme>` for a URL, the HTTP connection MUST be made  
313 over either SSL 3.0 (see [[SSLv3](#)]) or TLS 1.0 (see [[RFC2246](#)]) or any subsequent protocols that  
314 are backwards compatible with SSL 3.0 and/or TLS 1.0. Other security protocols MAY be used  
315 as long as they implement equivalent security measures.
- 316 3 Messages between providers MUST have their integrity protected, confidentiality MUST be  
317 ensured and the recipient MUST authenticate the sender.
- 318 4 Providers MUST use secure transport (`https`) to achieve confidentiality and integrity protection.  
319 The initiator of the secure connection MUST authenticate the server using server-side X.509  
320 certificates.
- 321 5 The authenticated identity of an identity provider MUST be securely available to a Principal  
322 before the Principal presents his/her personal authentication data to that identity provider.

- 323 6 For signing and verification of protocol messages, [LibertyProtSchema], identity providers and  
324 service providers SHOULD use certificates and private keys that are distinct from the  
325 certificates and private keys applied for SSL or TLS channel protection. Certificates and private  
326 keys MUST be suitable for long-term signatures.
- 327 7 In transactions between service providers and identity providers, requests MUST be protected  
328 against replay, and received responses MUST be checked for correct correspondence with issued  
329 requests. (Note: Other steps may intervene between the issuance of a request and its eventual  
330 response within a multistep transaction involving redirections.) Additionally, time-based  
331 assurance of freshness MAY be provided.
- 332 8 Each service provider within a circle of trust MUST be configured to enable identification of the  
333 identity providers whose authentications it will accept, and each identity provider MUST be  
334 configured to enable identification of the service providers it intends to serve. (Note: The format  
335 of this configuration is a local matter and could, for example, be represented as lists of names or  
336 as sets of X.509 certificates of other circle of trust members).
- 337 9 Circle of trust bilateral agreements on selecting certificate authorities, obtaining X.509  
338 credentials, establishing and managing trusted public keys, and tracking lifecycles of  
339 corresponding credentials are assumed and not in scope for this specification.
- 340 10 The <scheme> of the URL for SOAP endpoints MUST be https.
- 341 11 All SOAP message exchanges MUST adhere to the SOAP protocol binding for Liberty (see  
342 2.1).

### 343 **3.1.1 User Agent**

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

- 346 • HTTP 1.0 (see [RFC1945]) or HTTP 1.1 (see [RFC2616]).
- 347 • SSL 3.0 (see [SSLv3]) or TLS 1.0 (see [RFC2246]) or any subsequent protocols which are  
348 backwards compatible with SSL 3.0 and/or TLS 1.0 either directly or via a proxy (for  
349 example, a WAP gateway).
- 350 • Minimum maximum URL length of 256 bytes.

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

### 354 **3.1.2 Formatting and Encoding of Protocol Messages**

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

#### 357 **3.1.2.1 Encoding URL-embedded Messages**

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

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

- 362 • The `<query>` component parameter name MUST be the Liberty XML protocol message  
363 element or attribute name.
- 364 • The `<query>` component parameter value MUST be the value of the Liberty XML protocol  
365 message element or attribute value.
- 366 • When the original message element has multiple values, the value of the `<query>`  
367 component parameter MUST be a space-delimited list.
- 368 • Some of the referenced protocol message elements and attributes are optional. If an optional  
369 element or attribute does not appear in the original Liberty XML protocol message, then the  
370 corresponding data item MUST be omitted from the URL encoded message.
- 371 • URLs appearing in the URL-encoded message SHOULD NOT exceed 80 bytes in length  
372 (including %-escaping overhead). Likewise, the `<lib:RelayState>` data value SHOULD  
373 NOT exceed 80 bytes in length.

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

- 377 • Include the signature algorithm identifier as a new `<query>` component parameter named  
378 `SigAlg`, but omitting the signature.
- 379 • Sign the string containing the URL-encoded message. The string to be signed MUST include  
380 only the `<query>` part of the URL (that is, everything after `?` and before `&Signature=`).  
381 Any required URL-escaping MUST be done before signing.
- 382 • Encode the signature using base64 (see [\[RFC2045\]](#)).
- 383 • Add the base64-encoded signature to the encoded message as a new data item named  
384 `Signature`.

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

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

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

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

- 392 • DSAwithSHA1 — <http://www.w3.org/2000/09/xmlldsig#dsa-sha1>
- 393 • RSAwithSHA1 — <http://www.w3.org/2000/09/xmlldsig#rsa-sha1>

### 394 3.1.2.1.1 Size Limitations

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



### 402 3.1.2.1.2 URL-encoded <lib:AuthnRequest>

403 The original <lib:AuthnRequest> message:

```
404 <lib:AuthnRequest
405   RequestID="[RequestID]"
406   MajorVersion="[MajorVersion]"
407   MinorVersion="[MinorVersion]"
408   IssueInstant="[IssueInstant]">
409   <lib:ProviderID> [ProviderID] </lib:ProviderID>
410   <lib:ForceAuthn> [ForceAuthn] </lib:ForceAuthn>
411   <lib:IsPassive> [IsPassive] </lib:IsPassive>
412   <lib:Federate> [Federate] </lib:Federate>
413   <lib:ProtocolProfile> [ProtocolProfile] </lib:ProtocolProfile>
414   <lib:AuthnContext>
415     <lib:AuthnContextStatementRef>
416       [AuthnContextStatementRef]
417     </lib:AuthnContextStatementRef>
418   </lib:AuthnContext>
419   <lib:RelayState> [RelayState] </lib:RelayState>
420   <lib:AuthnContextComparisonType>
421     [AuthnContextComparisonType]
422   </lib:AuthnContextComparisonType>
423 </lib:AuthnRequest>
```

424

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

427 RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID,  
428 ForceAuthn, IsPassive, Federate, ProtocolProfile,  
429 AuthnContextStatementRef, AuthnContextClassRef, RelayState

- 430 • Maximum size: 748 bytes + 81 \* number of AuthnContextClassRefs

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

```
432 http://idp.example.com/authn?RequestID=RMvY34pg%2FV9aGJ5yw0HL0AcjccqQF&MajorVersion=1&MinorVersion
433 =0&IssueInstant=2002-05
434 15T00%3A58%3A19&ProviderID=http%3A%2F%2Fsp.example.com%2Fliberty%2F&ForceAuthn=true&IsPassive=fal
435 se&Federate=true&ProtocolProfile=http%3A%2F%2Fprojectliberty.org%2Fprofiles%2Fbrws-
436 post&AuthnContextClassRef=http%3A%2F%2Fprojectliberty.org%2Fauthnctx%2Fprofiles%2Fpassword-over-
437 HTTP&RelayState=03mhakSms5tmQ0WRDCEzpf7BNcywZa75FwIcSSEFvbkoFxaQHCuNnc5yChIdDlWc7JbV9Xbw3avRBK7VF
438 sPl2X&SigAlg=http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23rsa-
439 sha1&Signature=EoD8bNr2jEOe%2Fumon6oU%2FZGIIIF7gbJAe4MLUUMrD%2BPP7P8Yf3gfdZG2qPJdNAJkzVHGfO8W8DzpQ
440 %0D%0AsDTTd5VP9MLPcvxbFQoF0CJmVL26cPsuc54q7ourcH0jJ%2F2UkDq4DA1YLZ5kPIg%2BtrykgLz0U%2BS%0D%0ANqp
441 NHkjh6W3YkGv7RBs%3D
```

### 442 3.1.2.1.3 URL-Encoded <lib:FederationTerminationNotification>

443 The original <lib:FederationTerminationNotification> message:

```
444 <lib:FederationTerminationNotification ...
445   RequestID="[RequestID]"
446   MajorVersion="[MajorVersion]"
447   MinorVersion="[MinorVersion]"
448   IssueInstant="[IssueInstant]">
449   <lib:ProviderID> [ProviderID] </lib:ProviderID>
450   <saml:NameIdentifier NameQualifier="[NameQualifier]"
451     Format="[NameFormat]"
452     [NameIdentifier]
453   </saml:NameIdentifier>
454 </lib:FederationTerminationNotification>
```

455

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

458 RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID,  
459 NameQualifier, NameFormat, NameIdentifier

#### 460 3.1.2.1.4 URL-Encoded <lib:LogoutNotification>

461 The original <lib:LogoutNotification> message:

```
462 <lib:LogoutNotification ...  
463   RequestID="[RequestID]"  
464   MajorVersion="[MajorVersion]"  
465   MinorVersion="[MinorVersion]"  
466   IssueInstant="[IssueInstant]">  
467     <lib:ProviderID>[ProviderID]</lib:ProviderID>  
468     <saml:NameIdentifier NameQualifier="[NameQualifier]"  
469       Format="[NameFormat]">  
470       [NameIdentifier]  
471     </saml:NameIdentifier>  
472 </lib:LogoutNotification>
```

473

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

476 RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID,  
477 NameQualifier, NameFormat, NameIdentifier

#### 478 3.1.3 Provider Metadata

479 The majority of the Liberty profiles defined in this document rely on metadata that specify the  
480 policies that govern the behavior of the service provider or identity provider. These provider  
481 metadata are typically shared out of band between an identity provider and a service provider prior to  
482 the exchange of Liberty protocol messages. The provider metadata relevant to each profile are listed  
483 in this document at the beginning of the profile category. Refer to [[LibertyProtSchema](#)] for a  
484 complete enumeration of the Liberty provider metadata elements and their associated schema.

#### 485 3.2 Single Sign-On and Federation Profiles

486 This section defines the profiles by which a service provider obtains an authentication assertion from  
487 an identity provider to facilitate single sign-on. Additionally, the single sign-on profiles can be used  
488 as a means of federating an identity from a service provider to an identity provider through the use of  
489 the <Federate> element in the <lib:AuthnRequest> protocol message as specified in  
490 [[LibertyProtSchema](#)].

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

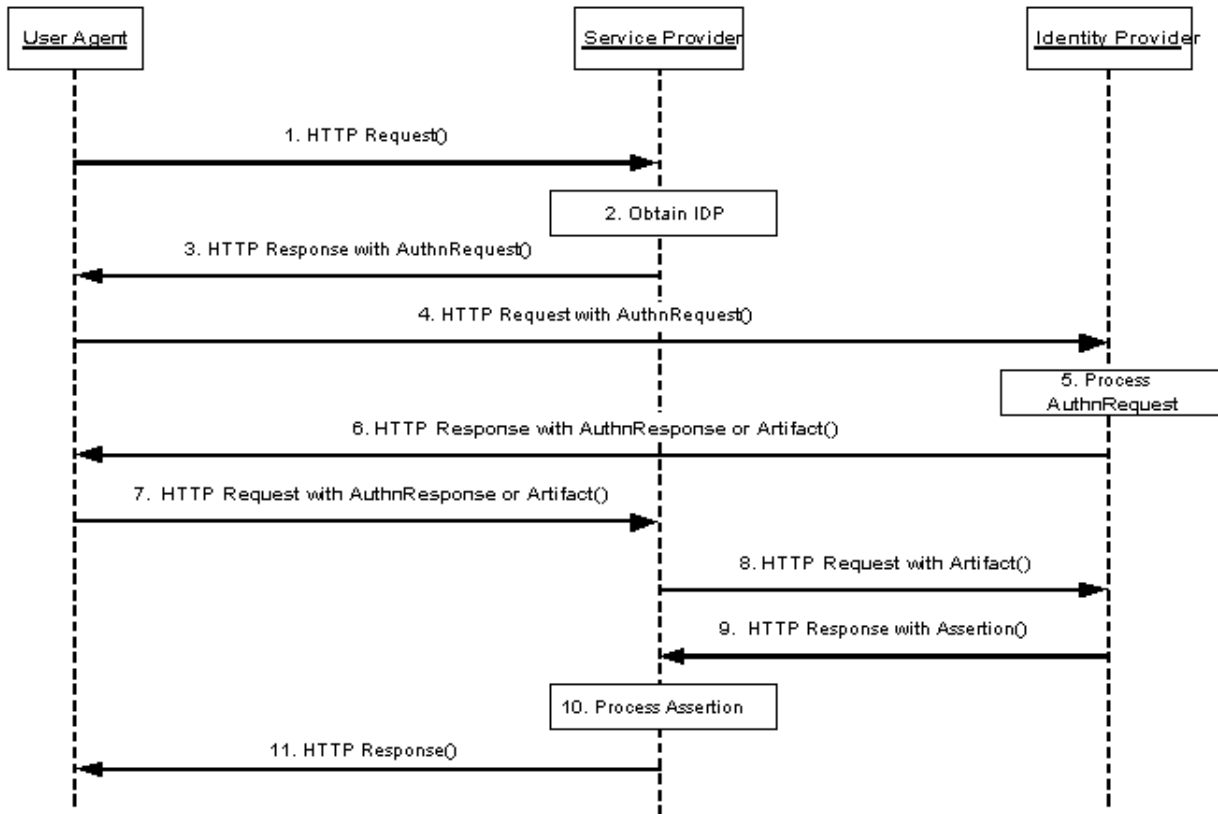
- 493 • ProviderID — Used to uniquely identify the service provider to the identity provider and  
494 is documented in these profiles as “service provider ID.”
- 495 • SingleSignOnServiceURL — The URL at the identity provider that the service provider  
496 should use when sending single sign-on and federation requests. It is documented in these  
497 profiles as “single sign-on service URL.”
- 498 • AssertionConsumerServiceURL — The URL at the service provider that an identity  
499 provider should use when sending single sign-on or federation responses. It is documented in  
500 these profiles as “assertion consumer service URL.”
- 501 • SOAPEndPoint — The SOAP endpoint location at the service provider or identity provider  
502 to which Liberty SOAP messages are sent.



503 **3.2.1 Common Interactions and Processing Rules**

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

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



511  
512 **Figure 1: Basic single sign-on profile**

513 **Step 1: HTTP Request**

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

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

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

521 The <query> component can be used to convey information about the originally requested  
522 resource at the service provider. It is RECOMMENDED that the <query> parameter be named

523 LRURL and its value be the URL originally requested by the user agent.

524

525 It is RECOMMENDED that the HTTP request be made over either SSL 3.0 (see [[SSLv3](#)]) or TLS  
526 1.0 (see [[RFC2246](#)]) to maintain confidentiality and message integrity in step 1.

## 527 **Step 2: Obtain Identity Provider**

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

## 532 **Step 3: HTTP Response with <AuthnRequest>**

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

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

## 536 **Step 4: HTTP Request with <AuthnRequest>**

537 In step 4, the user agent accesses the identity provider's single sign-on service URL with the  
538 <lib:AuthnRequest> information.

## 539 **Step 5: Processing <AuthnRequest>**

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

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

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

## 551 **Step 6: HTTP Response with <AuthnResponse> or Artifact**

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

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

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

558 If the identity provider discovers a syntax error due to the service provider or LECP or cannot proceed any  
559 further for other reasons (for example, unsupported Liberty version), the identity provider MUST return to the

560 LECP a `<lib:AuthnResponseEnvelope>` containing a `<lib:AuthnResponse>` with a  
561 `<samlp:Status>` indicating the desired error element as defined in [[LibertyProtSchema](#)].

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

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

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

### 566 **Step 8: HTTP Request with Artifact**

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

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

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

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

### 574 **Step 9: HTTP Response with Assertion**

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

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

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

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

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

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

### 594 **Step 10: Process Assertion**

595 In step 10, the service provider processes the `<saml:Assertion>` returned in the  
596 `<samlp:Response>` or `<lib:AuthnResponse>` protocol message to determine its validity and

597 how to respond to the Principal's original request. The signature on the <saml:Assertion> must  
598 be verified.

599 The service provider processing of the assertion MUST adhere to the rules defined in [[SAMLCore](#)]  
600 for things such as assertion <saml:Conditions> and <saml:Advice>.

601 The service provider MAY obtain authentication context information for the Principal's current  
602 session from the <lib:AuthnContext> element contained in the <saml:advice>. Similarly, the  
603 information in the <lib:RelayState> element MAY be obtained and used in further processing  
604 by the service provider.

## 605 **Step 11: HTTP Response**

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

## 608 **3.2.2 Liberty Browser Artifact Profile**

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

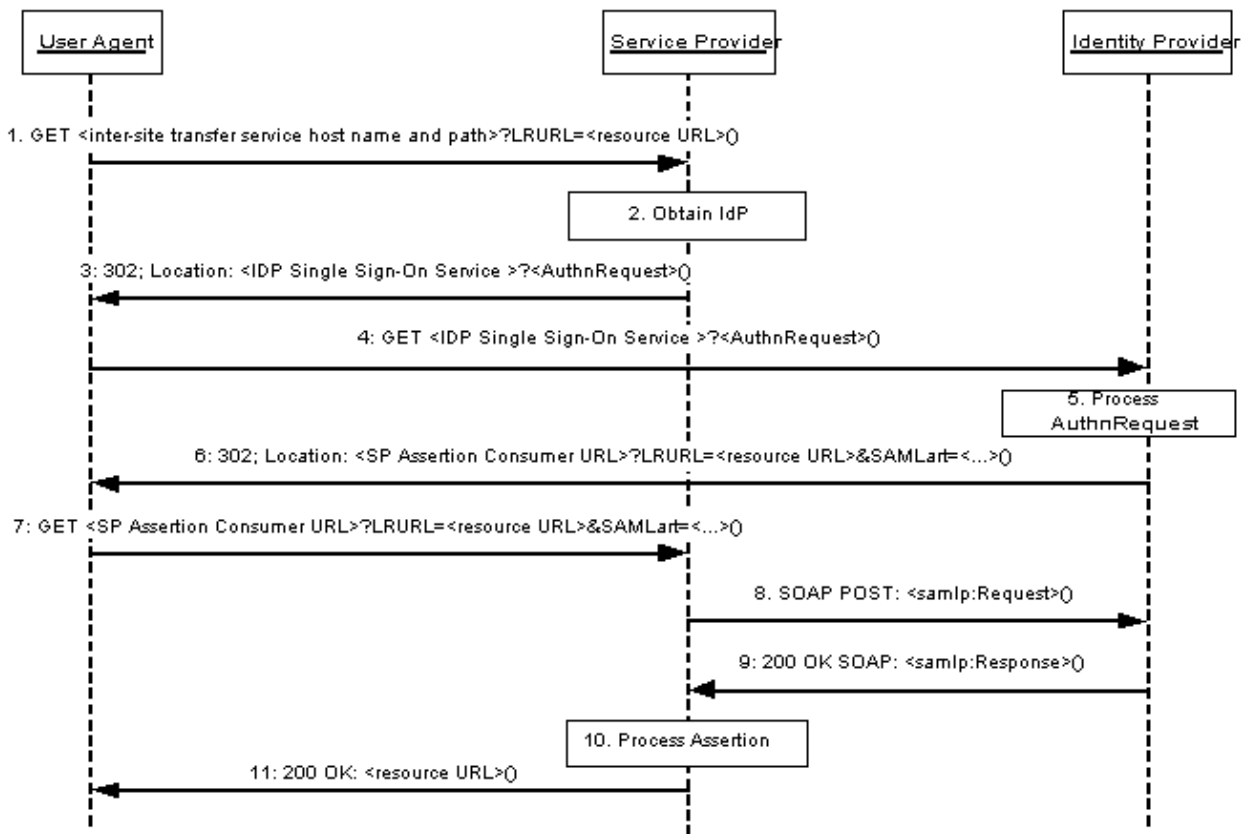
613 The following URI-based identifier MUST be used when referencing this specific profile (for  
614 example, <lib:ProtocolProfile> element of the <lib:AuthnRequest> message):

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

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

### 619 **3.2.2.1 Interactions**

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



621

622

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

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

625 When implementing this profile, all processing rules defined in 3.2.1 for the single sign-on profiles  
 626 MUST be followed. Additionally, the following rules MUST be observed as they relate to steps 3, 6  
 627 and 7:

628 **Step 3: Single sign on Service with <AuthnRequest>**

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

631 The redirection MUST adhere to the following rules:

- 632 • The Location HTTP header MUST be set to the identity provider’s single sign-on service  
 633 URL.
- 634 • The identity provider’s single sign-on service URL MUST specify https as the URL  
 635 scheme; if another scheme is specified, the service provider MUST NOT redirect to the  
 636 identity provider.

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

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

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

650       The HTTP response MUST take the following form:

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

655       where

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

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

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

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

## 661       **Step 6: Redirecting to the Service Provider**

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

664       The redirection MUST adhere to the following rules:

- 665       • The Location HTTP header MUST be set to the service provider's assertion consumer  
666       service URL, the value of which was determined based upon the `<lib:ProviderID>`  
667       element of the `<lib:AuthnRequest>` message.
- 668       • The service provider's assertion consumer service URL MUST specify `https` as the URL  
669       scheme; if another scheme is specified, the identity provider MUST NOT redirect to the  
670       service provider.
- 671       • The Location HTTP header MUST include a `<query>` parameter `SAMLart`, the value of  
672       which is the SAML artifact on success or on failure. In the case of failure, the status  
673       information will be communicated in the `<samlp:Response>` returned in step 9.  
674       Additionally, if the `<lib:AuthnRequest>` processed in step 5 included a value for the  
675       `<lib:RelayState>` element, then a parameter named `LRURL` with a value set to the value  
676       of the `<lib:RelayState>` element MUST be included in the `<query>` component.

677       The HTTP response MUST take the following form:

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

682       where

683       `<Service Provider assertion consumer service URL>`

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

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

687 At least one SAML artifact **MUST** be included in the <query> component. A single LRURL  
688 **MUST** be included if a value for <RelayState> was provided in the <lib:AuthnRequest>.

689 If more than one SAML artifact is included the <query> component, all artifacts **MUST** have  
690 the same IdentityProviderID.

## 691 Step 7: Accessing the Assertion Consumer Service

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

### 694 3.2.2.2 Artifact Format

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

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

698

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

```
703 TypeCode := 0x0003  
704 RemainingArtifact := IdentityProviderSuccinctID AssertionHandle  
705 IdentityProviderSuccinctID:= 20-byte sequence  
706 AssertionHandle := 20-byte_sequence
```

707

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

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

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

- 721 • Each identity provider selects a single identification URL, corresponding to the provider  
722 metadata element ProviderID specified in [[LibertyProtSchema](#)].
- 723 • The identity provider constructs the IdentityProviderSuccinctID component of the  
724 artifact by taking the SHA-1 hash of the identification URL. This 20-byte binary value  
725 corresponds to the hex-encoded value in the provider metadata element  
726 ProviderSuccinctID specified in [[LibertyProtSchema](#)]. Note that the  
727 IdentityProviderSuccinctID value, used to construct the artifact, is not encoded in  
728 hexadecimal, unlike the ProviderSuccinctID value used in the provider metadata.



- 729       • The identity provider SHOULD employ best effort to ensure the uniqueness of the generated  
730       IdentityProviderSuccinctID.

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

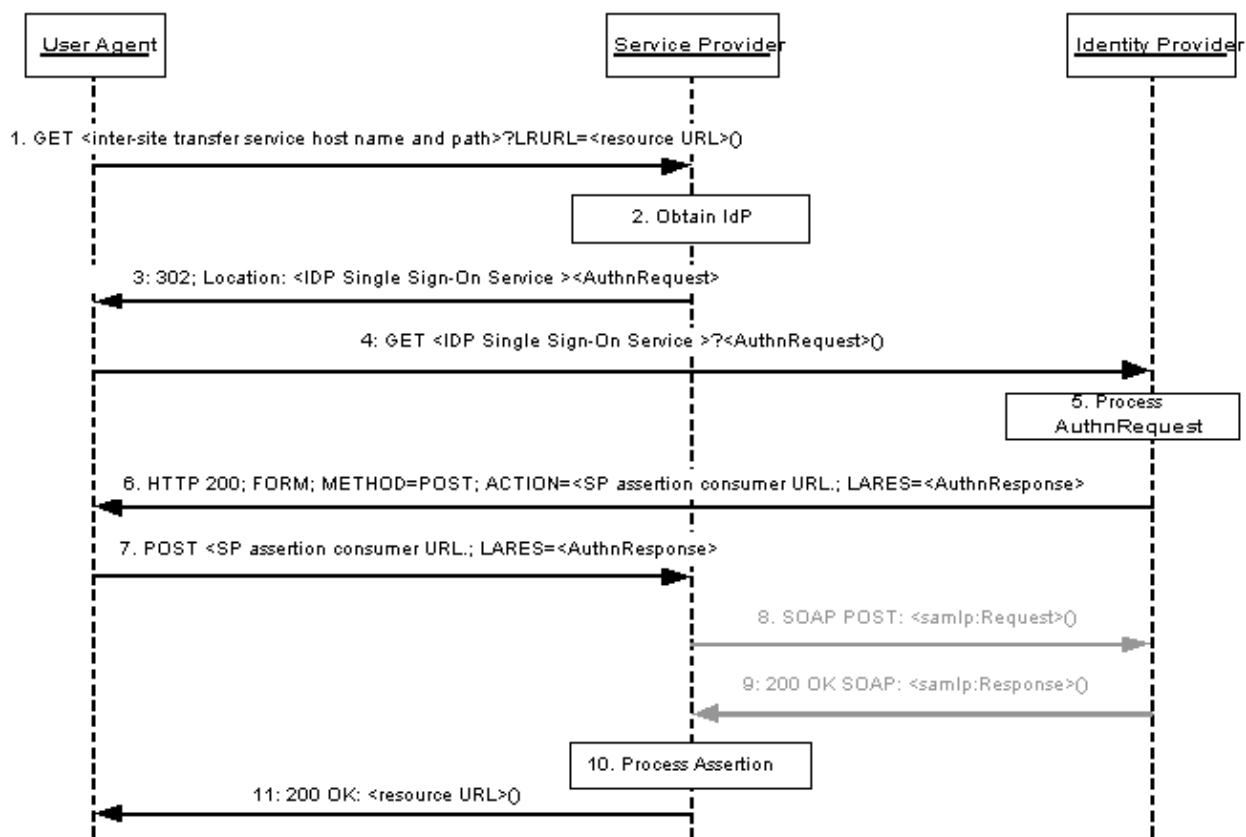
### 734 3.2.3 Liberty Browser POST Profile

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

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

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

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



744

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

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



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

751 **Step 3: Single Sign On Service with <AuthnRequest>:**

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

754 The redirection MUST adhere to the following rules:

- 755 • The Location HTTP header MUST be set to the identity provider's single sign-on service  
756 URL.
- 757 • The identity provider's single sign-on service URL MUST specify https as the URL  
758 scheme; if another scheme is specified, the service provider MUST NOT redirect to the  
759 identity provider.

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

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

765 Note: The <lib:RelayState> element of the <lib:AuthnRequest> message  
766 can be used by the service provider to help maintain state information during the  
767 single sign-on and federation process. For example, the originally requested resource  
768 (that is, LRURL in step 1) could be stored as the value for the <lib:RelayState>  
769 element, which would then be returned to the service provider in the  
770 <lib:AuthnResponse> in step 7. The service provider could then use this  
771 information to know how to formulate the HTTP response to the user agent in  
772 step 11.

773 The HTTP response MUST take the following form:

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

778 where

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

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

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

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

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

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

787 The form MUST be constructed so that it requests a POST to the service provider's assertion  
788 consumer URL with form contents that contain the field LARES with the value being the

789 <lib:AuthnResponse> protocol message as defined in [[LibertyProtSchema](#)]. The

790 <lib:AuthnResponse> MUST be encoded by applying a base64 transformation (refer to

791 [[RFC2045](#)) to the <lib:AuthnResponse> and all its elements. The service provider's assertion  
792 consumer service URL used as the target of the form POST MUST specify `https` as the URL  
793 scheme; if another scheme is specified, it MUST be treated as an error by the identity provider.

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

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

### 798 **Step 7: Posting the Form Containing the <AuthnResponse>**

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

### 801 **3.2.4 Liberty WML POST Profile**

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

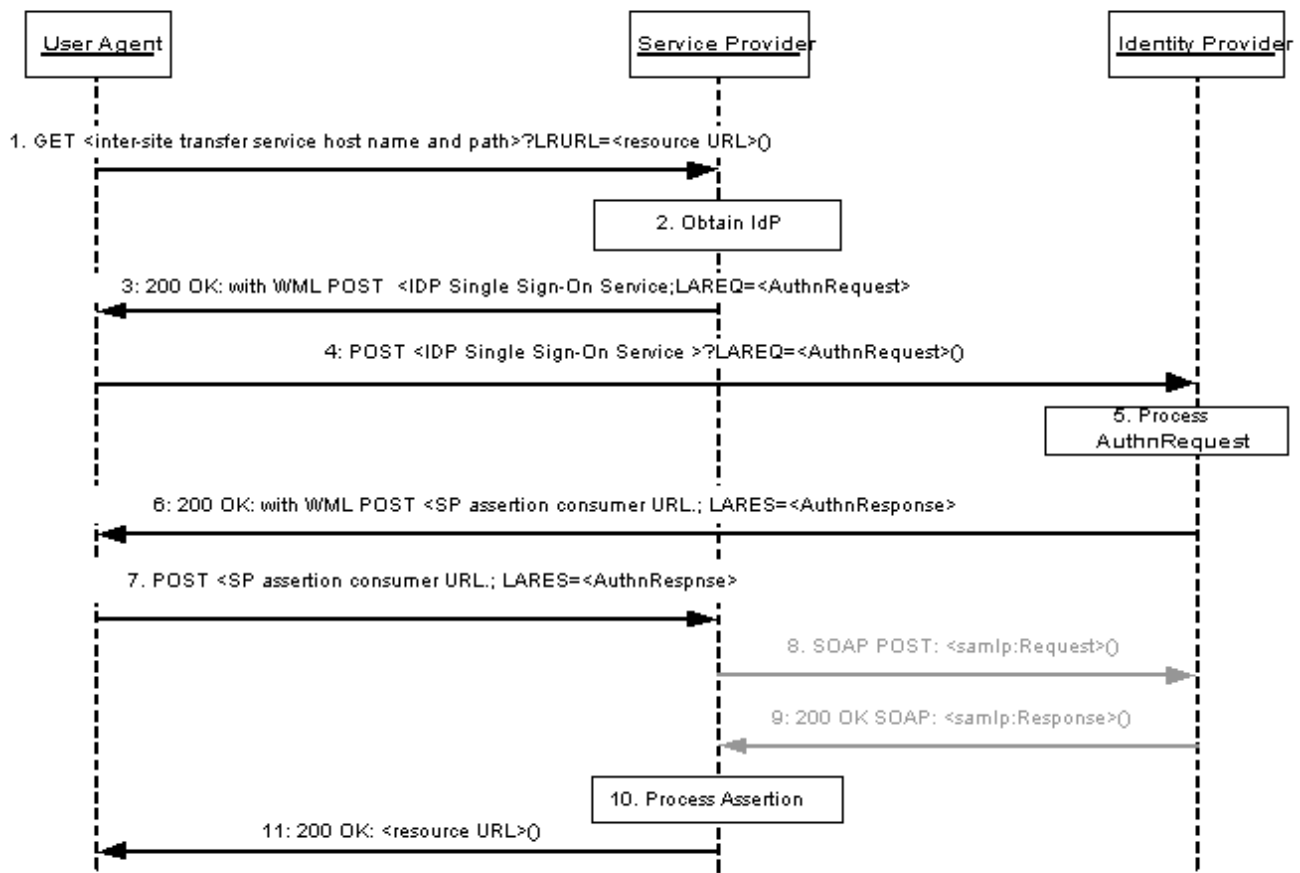
805 The following URI-based identifier MUST be used when referencing this specific profile (for  
806 example, <lib:ProtocolProfile> element of the <lib:AuthnRequest> message)

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

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

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

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



817

818

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

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

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

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

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

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

833 Note: One method for seamlessly instructing the user agent to POST the  
834 <lib:AuthnRequest> is to include a WML deck (see Chapter 17 in [HTML4])

835 within the HTTP 200 response. The following is an example of how the WML code  
836 could be structured:

```
837 ...  
838 <wml>  
839 <card id="redirect" title="Log In">  
840   <onenterforward>  
841     <go method="post" href="<Identity Provider Single Sign-On service URL>" >  
842       <postfield name="LAREQ" Value="(<lib:AuthnRequest>)" />  
843   </go>  
844   </onenterforward>  
845   <onenterbackward>  
846     <prev/>  
847   </onenterbackward>  
848   <p>  
849     Contacting IdP. Please wait...  
850   </p>  
851 ...  
852 </card>  
853 ...  
854 </wml>
```

855

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

#### 861 **Step 4: HTTP Request with `<AuthnRequest>`**

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

#### 864 **Step 6: HTTP Response with `<AuthnResponse>`**

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

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

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

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

#### 879 **Step 7: HTTP POST with `<AuthnResponse>`**

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

### 882 3.2.5 Liberty-Enabled Client and Proxy Profile

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

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

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

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

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

#### 899 3.2.5.1 Liberty-Enabled Indications

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

```
903 Liberty-Enabled = "Liberty-Enabled" ":" LIB_Version [" ,"  
904 1#Extension]  
905 LIB_Version = "LIBV" "=" 1*absoluteURI  
906 ; any spaces or commas in the absoluteURI MUST be escaped as  
907 defined in section 2.4 of [RFC 2396]  
908 Extension = ExtName "=" ExtValue  
909 ExtName = ([ "." host ] | <any field-value but ".", ",", " or "=">) <any  
910 field-value but "=" or ",", ">  
911 ExtValue = <any field-value but ",", ">
```

912  
913 The comment, field-value, and product productions are defined in [[RFC 2616](#)]. `LIB_Version`  
914 identifies the versions of the Liberty specifications that are supported by this LECP. Each version is  
915 identified by a URI. Service providers or identity providers receiving a Liberty-Enabled header  
916 MUST ignore any URIs listed in the `LIB_Version` production that they do not recognize. All  
917 LECPs compliant with this specification MUST send out, at minimum, the URI  
918 `http://projectliberty.org/specs/v1` as a value in the `LIB_Version` production. The  
919 ordering of the URIs in the `LIB_Version` header is meaningful; therefore, service providers and  
920 identity providers are encouraged to use the first version in the list that they support. Supported  
921 Liberty versions are not negotiated between the LECP and the service provider. The LECP simply  
922 advertises what version it does support, and the service provider MUST return the response for the  
923 corresponding version as defined in step 3 below.

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

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

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

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

934

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

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

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

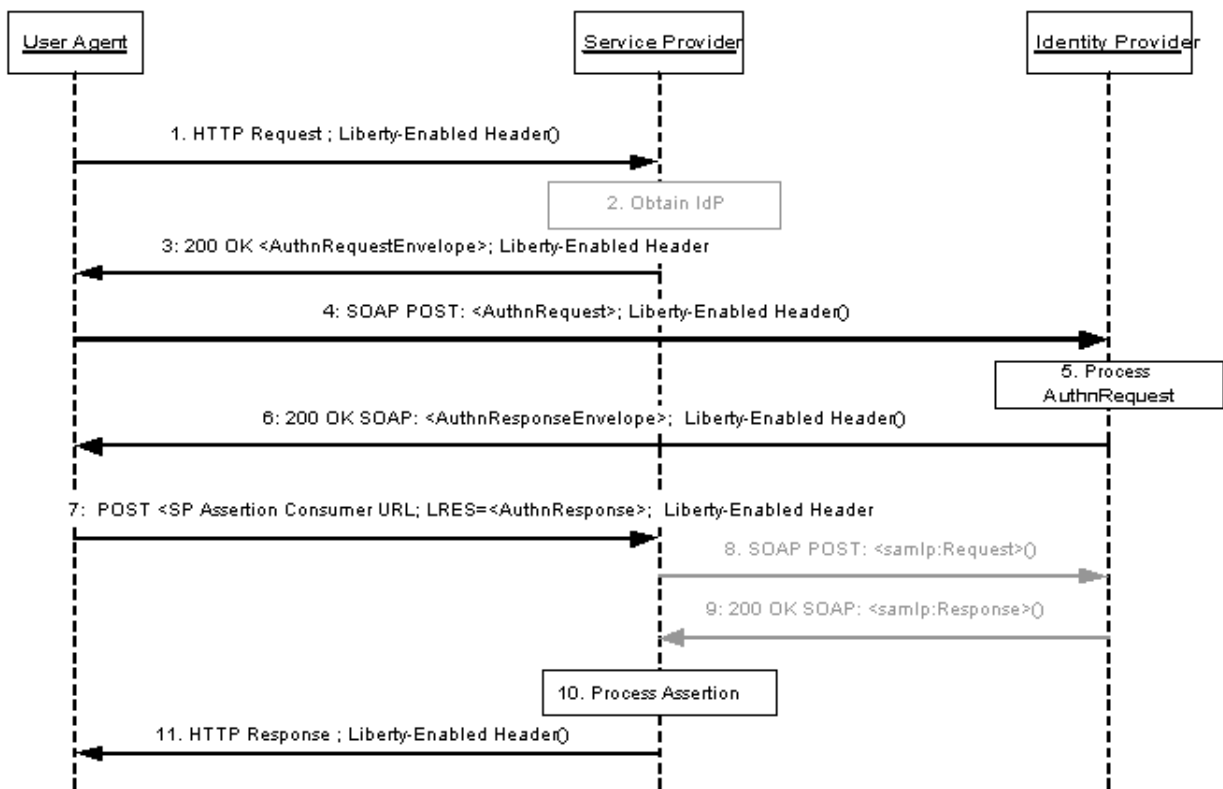
950 Note: The reason for adding information to the User-Agent header is to allow for  
951 Liberty-enabled client products that must rely on a platform that cannot be instructed  
952 to insert new headers in each HTTP request.

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

### 956 3.2.5.2 Interactions

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





958

959

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

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

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

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

### 971 **Step 1: Accessing the Service Provider**

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

974 The HTTP request MUST contain only one Liberty-Enabled header. Hence if a proxy receives a  
 975 HTTP request that contains a Liberty-Enabled header, it MUST NOT add another Liberty-Enabled  
 976 header. However, a proxy MAY replace the Liberty-Enabled header. A proxy that replaces or adds a  
 977 Liberty-Enabled header MUST process <lib:AuthnRequest> messages as defined in steps 3 and  
 978 4 as well as <lib:AuthnResponse> messages as specified in steps 6 and 7.

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

### 981 **Step 3: HTTP Response with <AuthnRequest>**

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

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

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

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

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

1000 The service provider MUST specify a URL for receiving <AuthnResponse> elements, locally  
1001 generated by the intermediary, by including the <lib:AssertionConsumerServiceURL>  
1002 element in the <lib:AuthnRequestEnvelope>.

1003 The following example demonstrates the usage of the <lib:AuthnRequestEnvelope>:

```
1004 <?xml version="1.0" ?>  
1005 <lib:AuthnRequestEnvelope xmlns:lib="http://projectliberty.org/schemas/core/2002/12/">  
1006   <lib:AssertionConsumerServiceURL>  
1007     https://service-provider.com/LibertyLogin  
1008   </lib:AssertionConsumerServiceURL>  
1009   <lib:IDPList >  
1010     . . . IdP list goes here . . .  
1011   </lib:IDPList>  
1012   <lib:AuthnRequest >  
1013     . . . AuthnRequest goes here . . .  
1014   </lib:AuthnRequest>  
1015 </lib:AuthnRequestEnvelope>
```

1017 If the service provider does not support the LECP-advertised Liberty version, the service provider  
1018 MUST return to the LECP an HTTP 501 response with the reason phrase “Unsupported Liberty  
1019 Version.”

1020 The responses in step 3 and step 6 SHOULD NOT be cached. To this end service providers and  
1021 identity providers SHOULD place both “Cache-Control: no-cache” and “Pragma: no-  
1022 cache” on their responses to ensure that the LECP and any intervening proxies will not cache the  
1023 response.



1024 **Step 4: HTTP Request with <AuthnRequest>**

1025 In step 4, the LECP determines the appropriate identity provider to use and then issues an HTTP  
1026 POST of the <lib:AuthnRequest> in the body of a SOAP message to the identity provider's  
1027 single sign-on service URL. The request MUST contain the same <lib:AuthnRequest> as was  
1028 received in the <lib:AuthnRequestEnvelope> from the service provider in step 3.

1029 Note: The identity provider list can be used by the LECP to create a user identifier to be  
1030 presented to the Principal. For example, the LECP could compare the list of the Principal's  
1031 known identities (and the identities of the identity provider that provides those identities)  
1032 against the list provided by the service provider and then only display the intersection.

1033 If the LECP discovers a syntax error due to the service provider or cannot proceed any further for  
1034 other reasons (for example, cannot resolve identity provider, cannot reach the identity provider, etc),  
1035 the LECP MUST return to the service provider a <lib:AuthnResponse> with a  
1036 <samlp:Status> indicating the desired error element as defined in [[LibertyProtSchema](#)]. The  
1037 <lib:AuthnResponse> containing the error status MUST be sent using a POST to the service  
1038 provider's assertion consumer service URL obtained from the  
1039 <lib:AssertionConsumerServiceURL> element of the <lib:AuthnRequestEnvelope>.  
1040 The POST MUST be a form that contains the field named LARES with its value the  
1041 <lib:AuthnResponse> protocol message as defined in [[LibertyProtSchema](#)] with formatting as  
1042 specified in 3.1.2.

1043 **Step 6: HTTP Response with <AuthnResponse>**

1044 In step 6, the identity provider responds to the <lib:AuthnRequest> by issuing an HTTP 200 OK  
1045 response. The response MUST contain a single <lib:AuthnResponseEnvelope> in the body of a  
1046 SOAP message with content as defined in [[LibertyProtSchema](#)].

1047 The identity provider MUST include the Liberty-Enabled HTTP header following the same  
1048 processing rules as defined in 3.2.5.1.

1049 The Content-Type MUST be set to `application/vnd.liberty-response+xml`.

1050 Note: Identity providers that wish to authenticate LECPs via name and password  
1051 exchange can do so by challenging the LECP using HTTP authentication  
1052 mechanisms. Because the structure of the LECP profile requires all communications  
1053 to occur over `https`, the name and password pair can be securely sent either over  
1054 HTTP basic or digest authentication.

1055 If the identity provider discovers a syntax error due to the service provider or LECP or cannot  
1056 proceed any further for other reasons (for example, unsupported Liberty version), the identity  
1057 provider MUST return to the LECP a <lib:AuthnResponseEnvelope> containing a  
1058 <lib:AuthnResponse> with a <samlp:Status> indicating the desired error element as defined  
1059 in [[LibertyProtSchema](#)].

1060 **Step 7: Posting the Form Containing the <AuthnResponse>**

1061 In step 7, the LECP issues an HTTP POST of the <lib:AuthnResponse> that was received in the  
1062 <lib:AuthnResonseEnvelope> SOAP response in step 6. The <lib:AuthnResponse> MUST  
1063 be sent using a POST to the service provider's assertion consumer service URL identified by the  
1064 <lib:AssertionConsumerServiceURL> element within the  
1065 <lib:AuthnResponseEnvelope> *obtained from the identity provider in step 6*. The POST MUST

1066 be a form that contains the field `LARES` with the value being the `<lib:AuthnResponse>` protocol  
1067 message as defined in [[LibertyProtSchema](#)]. The `<lib:AuthnResponse>` MUST be encoded by  
1068 applying a base64 transformation (refer to [[RFC2045](#)]) to the `<lib:AuthnResponse>` and all its  
1069 elements. The service provider's assertion consumer service URL used as the target of the form  
1070 POST MUST specify `https` as the URL scheme; if another scheme is specified, it MUST be treated  
1071 as an error by the identity provider.

1072 If the LECP discovers an error (for example, syntax error in identity provider response), the LECP  
1073 MUST return to the service provider a `<lib:AuthnResponse>` with a `<samlp:Status>`  
1074 indicating the appropriate error element as defined in [[LibertyProtSchema](#)]. The  
1075 `<lib:AuthnResponse>` containing the error status MUST be sent using a POST to the service  
1076 provider's assertion consumer service URL. The POST MUST be a form that contains the field  
1077 named `LARES` with its value being the `<lib:AuthnResponse>` protocol message as defined in  
1078 [[LibertyProtSchema](#)] with formatting as specified 3.1.2. Any `<lib:AuthnResponse>` messages  
1079 created by the identity provider MUST not be sent to the service provider.

### 1080 3.3 Register Name Identifier Profiles

1081 This section defines the profile by which a service or identity provider MAY register or change a  
1082 name identifier for a Principal that the identity provider MUST use when communicating with the  
1083 service provider about that Principal. This message exchange is optional. When it is not used, the  
1084 identity provider will always communicate to the service provider about the Principal using the  
1085 `<IDPProvidedNameIdentifier>`. The service provider will communicate to the identity provider  
1086 using the `<SPProvidedNameIdentifier>` if it initiates the `RegisterNameIdentifier` profile  
1087 and can provide a `<SPProvidedNameIdentifier>`. The default is use of the  
1088 `<IDPProvidedNameIdentifier>`. Two profiles are specified: HTTP-Redirect-Based and  
1089 SOAP/HTTP-based.

1090 Either the identity provider or the service provider can initiate the register name identifier profile.  
1091 The available profiles are defined in 3.3.1 and 3.3.2, depending on whether the identity federation  
1092 termination notification process was initiated at the identity provider or service provider:

- 1093 • Register Name Identifier Initiated at Identity Provider
  - 1094 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate between
  - 1095 the identity provider and the service provider.
  - 1096 • **SOAP/HTTP-Based:** Relies on a SOAP call from the identity provider to the
  - 1097 service provider.
- 1098 • Register Name Identifier Initiated at Service Provider
  - 1099 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate between
  - 1100 the service provider and the identity provider.
  - 1101 • **SOAP/HTTP-Based:** Relies on a SOAP call from the service provider to the
  - 1102 identity provider.

1103 The interactions and processing rules for the SOAP/HTTP-based and HTTP-redirect-based profiles  
1104 are essentially the same regardless of whether register name identifier was initiated at the service  
1105 provider or at the identity provider, just the message flow directions are reversed.

1106 The register name identifier profiles make use of the following metadata elements, as defined in  
1107 [[LibertyProtSchema](#)]:

- 1108       • `RegisterNameIdentifierProtocolProfile`: The service provider's preferred  
1109       register name identifier profile, which should be used by the identity provider when  
1110       registering a new identifier. This would specify the URI based identifier for one of the  
1111       IDP Initiated register name identifier profiles.
- 1112       • `RegisterNameIdentifierServiceURL`: The URL used for user-agent-based  
1113       Register Name Identifier Protocol profiles.
- 1114       • `RegisterNameIdentifierServiceReturnURL`: The provider's redirecting URL for  
1115       use after HTTP name registration has taken place.
- 1116       • `SOAPEndpoint`: The SOAP endpoint location at the service provider or identity  
1117       provider to which Liberty SOAP messages are sent.

### 1118   **3.3.1     Register Name Identifier Initiated at Identity Provider**

1119   An identity provider MAY change the `<IDPProvidedNameIdentifier>` it has assigned a  
1120   Principal and transmit that information to a service provider. The `<NameIdentifier>` MAY be  
1121   changed without changing any federations. The reason an identity provider MAY wish to change the  
1122   name identifier for a Principal are implementation dependent and outside the scope of this  
1123   specification. Changing the `<NameIdentifier>` MAY be accomplished in either an HTTP-  
1124   Redirect-Based or SOAP/HTTP mode.

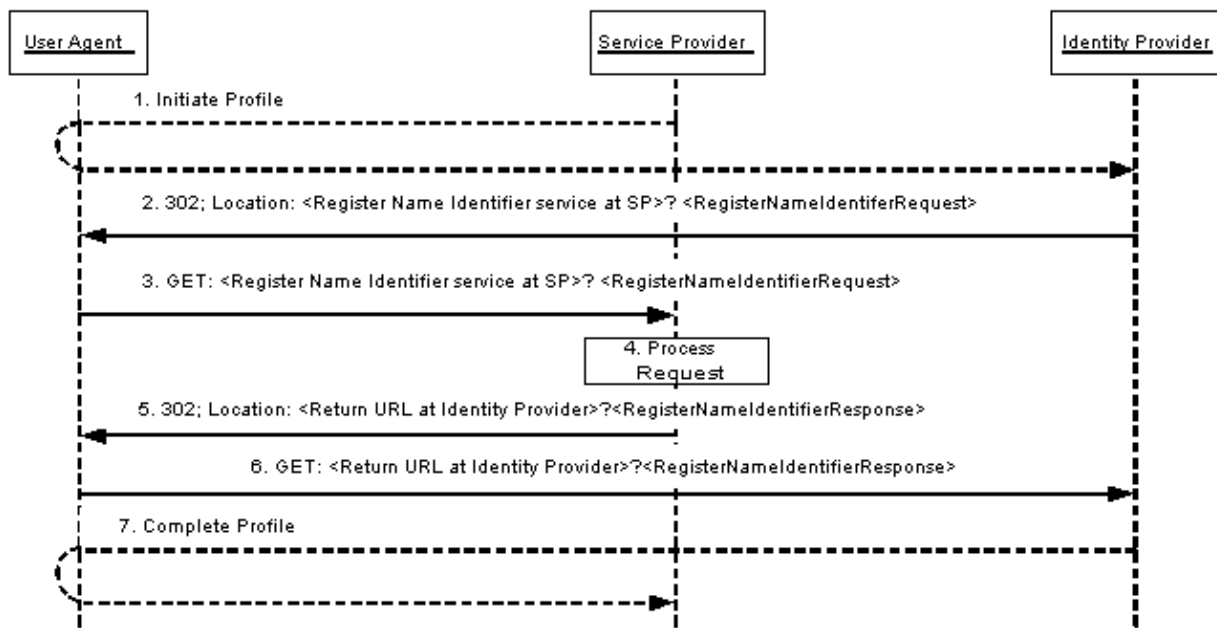
#### 1125   **3.3.1.1   HTTP-Redirect-Based Profile**

1126   A HTTP-redirect-based register name identifier profile cannot be self-initiated by an identity  
1127   provider, but MUST be a triggered by a message, such as an `<AuthnRequest>`. We note that we do  
1128   not normatively specify when and how the identity provider can initiate this profile—that is left to  
1129   the discretion of the identity provider. As an example, it MAY be triggered by a message, such as an  
1130   `<lib:AuthnRequest>`. In such a case when the identity provider decides to initiate this profile  
1131   after receiving an `<lib:AuthnRequest>`, it will insert this profile into the  
1132   `AuthnRequest/AuthnResponse` transaction to provide the new name identifier. The HTTP-  
1133   redirect-based profile relies on using HTTP 302 redirects to communicate register name identifier  
1134   messages from the identity provider to the service provider. The HTTP-Redirect Register Name  
1135   Identifier Profile (Figure 6) illustrates this transaction.

1136   The following URI-based identifier MUST be used when referencing this specific profile:

- 1137       • URI: <http://projectliberty.org/profiles/rni-idp-http>

1138   This URI identifier MUST be specified in the service provider metadata element  
1139   `RegisterNameIdentifierProtocolProfile` when the service provider intends to indicate to  
1140   the identity provider a preference for receiving register name identifier messages via a HTTP 302  
1141   redirect.



1142

1143

**Figure 6. Register Name Identifier Profile.**

1144 In an example scenario, the service provider makes an `<lib:AuthnRequest>` to the identity  
 1145 provider for authentication of the Principal's User Agent (step 1). The identity provider effects an  
 1146 `<lib:IDPProvidedNameIdentifier>` change in the service provider via a URL redirection. The  
 1147 profile is as follows:

1148 **Step 1: Initiate Profile**

1149 This interaction is not normatively specified as part of the profile, but shown for illustrative  
 1150 purposes.

1151 **Step 2: Redirecting to the Service Provider Register Name Identifier Service**

1152 In step 2, the identity provider redirects the user agent to the register name identifier service at the  
 1153 service provider.

1154 The redirection MUST adhere to the following rules:

- 1155 • The Location HTTP header MUST be set to the service provider's register name  
 1156 identifier service URL.
- 1157 • The service provider's registrater name identifier service URL MUST specify https as  
 1158 the URL scheme; if another scheme is specified, the identity provider MUST NOT  
 1159 redirect to the service provider.
- 1160 • The Location HTTP header MUST include a `<query>` component containing the  
 1161 `<lib:RegisterNameIdentifierRequest>` protocol message as defined in  
 1162 [[LibertyProtSchema](#)] with formatting as specified in 3.1.2.

1163 Note: Additionally, the URL-encoded  
 1164 `<lib:RegisterNameIdentifierRequest>` message MAY also include a

1165 parameter named RELAYSTATE with a value set to the URL (and/or other  
1166 information) to be used by the identity provider in the HTTP response to the user  
1167 agent at the completion of register name identifier in step 7.

1168 The HTTP response MUST take the following form:

```
1169 <HTTP-Version> 302 <Reason Phrase>  
1170 <other headers>  
1171 Location : https://<Service Provider Register Name Identifier service URL>?<query>  
1172 <other HTTP 1.0 or 1.1 components>
```

1173 where

1174 <Service Provider Register Name Identifier service URL>

1175 This element provides the host name, port number, and path components of the register name  
1176 identifier service URL at the service provider.

1177 <query>= ...<URL-encoded RegisterNameIdentifierRequest>...

1178 The <query> component MUST contain a single register name identifier request.

### 1179 Step 3: Accessing the Service Provider Register Name Identifier Service

1180 In step 3, the user agent accesses the service provider's register name identifier service URL with the  
1181 <lib:RegisterNameIdentifierRequest> information attached to the URL fulfilling the  
1182 redirect request.

### 1183 Step 4: Processing the Register Name Identifier Request

1184 In step 4, the service provider MUST process the <lib:RegisterNameIdentifierRequest>  
1185 according to the rules defined in [[LibertyProtSchema](#)].

1186 The service provider MAY remove the old name identifier after registering the new name identifier.

### 1187 Step 5: Redirecting to the Identity Provider return URL with the Register Name 1188 Identifier Response

1189 In step 5, the service provider's register name identifier service responds and redirects the user agent  
1190 back to identity provider using a return URL location specified in the  
1191 RegisterNameIdentifierReturnURL metadata element. If the URL-encoded <lib:  
1192 RegisterNameIdentifierRequest> message received in step 3 contains a parameter named  
1193 RELAYSTATE, then the service provider MUST include a <query> component containing the same  
1194 RELAYSTATE parameter and its value in its response to the identity provider.

1195 The redirection MUST adhere to the following rules:

- 1196 • The Location HTTP header MUST be set to the identity providers return URL specified  
1197 in the RegisterNameIdentifierReturnURL metadata element.
- 1198 • The identity provider's return URL MUST specify https as the URL scheme; if another  
1199 scheme is specified, the service provider MUST NOT redirect to the identity provider.
- 1200 • The Location HTTP header MUST include a <query> component containing the  
1201 <lib:RegisterNameIdentifierResponse> protocol message as defined in  
1202 [[LibertyProtSchema](#)] with formatting as specified in 3.1.2.

1203 The HTTP response MUST take the following form:

```
1204 <HTTP-Version> 302 <Reason Phrase>  
1205 <other headers>  
1206 Location : https://<Identity Provider Return URL >?<query>
```

1207 <other HTTP 1.0 or 1.1 components>

1208 where:

1209 <Identity Provider Return URL>

1210 This element provides the host name, port number, and path components of the return URL at the  
1211 identity provider.

1212 <query>= ...<URL-encoded RegisterNameIdentifierResponse>...

1213 The <query> component MUST contain a single register name identifier response. The <query>  
1214 component MUST contain the identical RELAYSTATE parameter and its value that was received in  
1215 the URL-encoded register name identifier message obtained in step 3. If no RELAYSTATE parameter  
1216 was provided in the step 3 message, then a RELAYSTATE parameter MUST NOT be specified in the  
1217 <query> component.

### 1218 **Step 6: Accessing the Identity Provider return URL with the Register Name Identifier** 1219 **Response**

1220 In step 6, the user agent accesses the identity provider's return URL location fulfilling the redirect  
1221 request.

### 1222 **Step 7: Complete profile**

1223 This concludes the initial sequence, which triggered the initiation of this profile.

#### 1224 **3.3.1.2 SOAP/HTTP-Based Profile**

1225 The following URI-based identifier MUST be used when referencing this specific profile:

- 1226 • URI: <http://projectliberty.org/profiles/rni-idp-soap>

1227 This URI identifier MUST be specified in the service provider metadata element  
1228 RegisterNameIdentifierProtocolProfile when the service provider intends to indicate to  
1229 the identity provider a preference for receiving register name identifier messages via SOAP over  
1230 HTTP.

1231 The steps involved in the SOAP/HTTP-based profile MUST utilize the SOAP binding for Liberty as  
1232 defined in 2.1. See Figure 7.



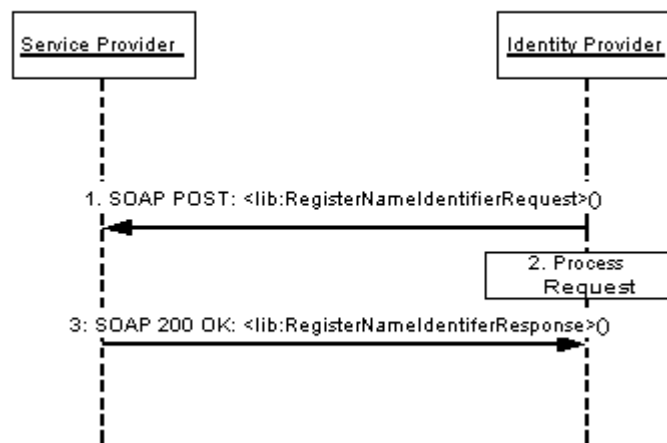


Figure 7: SOAP/HTTP-based profile for registering name identifiers

### Step 1: Request to Register Name Identifier

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

### Step 2: Process Request

Service provider records new `<IDPProvidedNameIdentifier>`.

### Step 3: Response to Register Name Identifier

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

## 3.3.2 Register Name Identifier Initiated at Service Provider

The service provider MAY effect a change in the Principal's name identifier using `<lib:SPProvidedNameIdentifier>`.

### 3.3.2.1 HTTP-Redirect-Based Profile

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

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

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

1257 A HTTP-redirect-based register name identifier profile can be self-initiated by a service provider to  
1258 change the `<lib:SPProvidedNameIdentifier>`. We note that we do not normatively specify  
1259 when and how the service provider can initiate this profile—that is left to the discretion of the service  
1260 provider. The HTTP-redirect-based profile relies on using HTTP 302 redirects to communicate  
1261 register name identifier messages from the service provider to the identity provider. The service  
1262 provider effects a `<lib:SPProvidedNameIdentifier>` change in the identity provider via a  
1263 URL redirection. For a discussion of the interactions and processing steps, refer to 3.3.1.1. When  
1264 reviewing that profile, interchange all references to service provider and identity provider in the  
1265 interaction diagram and processing steps 3-7.

### 1266 3.3.2.2 SOAP/HTTP-Based Profile

1267 The SOAP/HTTP-based profile relies on using SOAP over HTTP to communicate register name  
1268 identifier messages from the service provider to the identity provider. For a discussion of the  
1269 interactions and processing steps, refer to 3.3.1.2. When reviewing that profile, interchange all  
1270 references to service provider and identity provider in the interaction diagram and processing steps.

1271 The following URI-based identifier MUST be used when referencing this specific profile:

- 1272 • URI: <http://projectliberty.org/profiles/rni-sp-soap>

## 1273 3.4 Identity Federation Termination Notification Profiles

1274 The Liberty identity federation termination notification profiles specify how service providers and  
1275 identity providers are notified of federation termination (also known as defederation). Note: Other  
1276 means of federation termination are possible, such as federation expiration and termination of  
1277 business agreements between service providers and identity providers. These means of federation  
1278 termination are outside the scope of this specification.

1279 Identity federation termination can be initiated at either the identity provider or the service provider.  
1280 The Principal SHOULD have been authenticated by the provider at which identity federation  
1281 termination is being initiated. The available profiles are defined in 3.4.1 and 3.4.2, depending on  
1282 whether the identity federation termination notification process was initiated at the identity provider  
1283 or service provider:

- 1284 • **Federation Termination Notification Initiated at Identity Provider**

- 1285 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate between the  
1286 identity provider and the service provider.

- 1287 • **SOAP/HTTP-Based:** Relies on a SOAP call from the identity provider to the service  
1288 provider.

- 1289 • **Federation Termination Notification Initiated at Service Provider**

- 1290 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate between the  
1291 service provider and the identity provider.

- 1292 • **SOAP/HTTP-Based:** Relies on a SOAP call from the service provider to the identity  
1293 provider.

1294 The interactions and processing rules for the SOAP/HTTP-based and HTTP-redirect-based profiles  
1295 are essentially the same regardless of whether federation termination notification was initiated at the  
1296 service provider or at the identity provider.



1297 The identity federation termination notification profiles make use of the following metadata  
1298 elements, as defined in [[LibertyProtSchema](#)]:

- 1299 • `FederationTerminationServiceURL` — The URL at the service provider or identity  
1300 provider to which identity federation termination notifications are sent. It is documented in  
1301 these profiles as “federation termination service URL.”
- 1302 • `FederationTerminationServiceReturnURL` — The URL used by the service provider  
1303 or identity provider when redirecting the user agent at the end of the federation termination  
1304 notification profile process.
- 1305 • `FederationTerminationNotificationProtocolProfile` — Used by the identity  
1306 provider to determine which federation termination notification profile **MUST** be used when  
1307 communicating with the service provider.
- 1308 • `SOAPEndPoint` — The SOAP endpoint location at the service provider or identity provider  
1309 to which Liberty SOAP messages are sent.

### 1310 **3.4.1 Federation Termination Notification Initiated at Identity Provider**

1311 The profiles in 3.4.1.1 and 3.4.1.2 are specific to identity federation termination when initiated at the  
1312 identity provider. Effectively, when using these profiles, the identity provider is stating to the service  
1313 provider that it will no longer provide the Principal’s identity information to the service provider and  
1314 that the identity provider will no longer respond to any requests by the service provider on behalf of  
1315 the Principal.

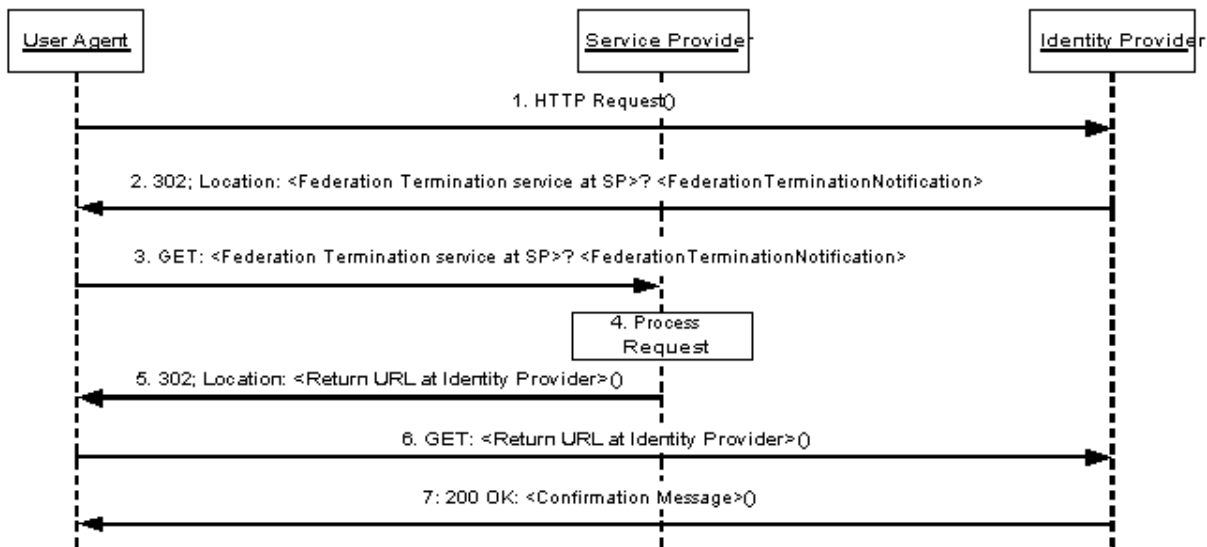
#### 1316 **3.4.1.1 HTTP-Redirect-Based Profile**

1317 The HTTP-redirect-based profile relies on using HTTP 302 redirect to communicate federation  
1318 termination notification messages from the identity provider to the service provider. See Figure 8.

1319 The following URI-based identifier **MUST** be used when referencing this specific profile:

1320 URI: `http://projectliberty.org/profiles/fedterm-idp-http`

1321 This URI identifier **MUST** be specified in the service provider metadata element  
1322 `FederationTerminationNotificationProtocolProfile` when the service provider intends  
1323 to indicate to the identity provider a preference for receiving federation termination notifications via  
1324 a HTTP 302 redirect.



1325

1326

**Figure 8: HTTP-redirect-based profile for federation termination**

1327 This profile description assumes the following preconditions:

- 1328 • The Principal’s identity at the service provider is federated with his/her identity at the
- 1329 identity provider.
- 1330 • The Principal has requested to the identity provider that the federation be terminated.
- 1331 • The Principal has authenticated with the identity provider.

1332 **Step 1: Accessing the Federation Termination Service**

1333 In step 1, the user agent accesses the identity federation termination service URL at the identity  
 1334 provider specifying the service provider with whom identity federation termination should occur.  
 1335 How the service provider is specified is implementation-dependent and, as such, is out of the scope  
 1336 of this specification.

1337 **Step 2: Redirecting to the Service Provider**

1338 In step 2, the identity provider’s federation termination service URL responds and redirects the user  
 1339 agent to the federation termination service at the service provider.

1340 The redirection MUST adhere to the following rules:

- 1341 • The Location HTTP header MUST be set to the service provider’s federation termination
- 1342 service URL.
- 1343 • The service provider’s federation termination service URL MUST specify `https` as the
- 1344 URL scheme; if another scheme is specified, the identity provider MUST NOT redirect to
- 1345 the service provider.
- 1346 • The Location HTTP header MUST include a `<query>` component containing the
- 1347 `<lib:FederationTerminationNotification>` protocol message as defined in
- 1348 [[LibertyProtSchema](#)] with formatting as specified in 3.1.2.

1349 Note: Additionally, the URL-encoded  
1350 `<lib:FederationTerminationNotification>` message MAY also include a  
1351 parameter named RELAYSTATE with a value set to the URL (and/or other  
1352 information) to be used by the identity provider in the HTTP response to the user  
1353 agent at the completion of federation termination in step 7.

1354 The HTTP response MUST take the following form:

```
1355 <HTTP-Version> 302 <Reason Phrase>  
1356 <other headers>  
1357 Location : https://<Service Provider Federation Termination service URL>?<query>  
1358 <other HTTP 1.0 or 1.1 components>
```

1359 where

1360 `<Service Provider Federation Termination service URL>`

1361 This element provides the host name, port number, and path components of the federation  
1362 termination service URL at the service provider.

1363 `<query>= ...<URL-encoded FederationTerminationNotification>...`

1364 The `<query>` component MUST contain a single terminate federation request.

### 1365 **Step 3: Accessing the Service Provider Federation Termination Service**

1366 In step 3, the user agent accesses the service provider's federation termination service URL with the  
1367 `<lib:FederationTerminationNotification>` information attached to the URL fulfilling the  
1368 redirect request.

### 1369 **Step 4: Processing the Notification**

1370 In step 4, the service provider MUST process the

1371 `<lib:FederationTerminationNotification>` according to the rules defined in  
1372 [[LibertyProtSchema](#)].

1373 The service provider MAY remove any locally stored references to the name identifier it received  
1374 from the identity provider in the `<lib:FederationTerminationNotification>`.

### 1375 **Step 5: Redirecting to the Identity Provider Return URL**

1376 In step 5, the service provider's federation termination service responds and redirects the user agent  
1377 back to identity provider using a return URL location specified in the

1378 `FederationTerminationServiceReturnURL` metadata element. If the URL-encoded

1379 `<lib:FederationTerminationNotification>` message received in step 3 contains a

1380 parameter named RELAYSTATE, then the service provider MUST include a `<query>` component  
1381 containing the same RELAYSTATE parameter and its value in its response to the identity provider.

1382 No success or failure message should be conveyed in this HTTP redirect. The sole purpose of this  
1383 redirect is to return the user agent to the identity provider where the federation termination process  
1384 began.

1385 The HTTP response MUST take the following form:

```
1386 <HTTP-Version> 302 <Reason Phrase>  
1387 <other headers>  
1388 Location : https://<Identity Provider Return URL >?<query>  
1389 <other HTTP 1.0 or 1.1 components>
```

1390 where

1391 `<Identity Provider Return URL>`

1392 This element provides the host name, port number, and path components of the return URL at the

1393 identity provider.  
1394 <query>= ...RELAYSTATE=<...>  
1395 The <query> component MUST contain the identical RELAYSTATE parameter and its value that  
1396 was received in the URL-encoded federation termination message obtained in step 3. If no  
1397 RELAYSTATE parameter was provided in the step 3 message, then a RELAYSTATE parameter  
1398 MUST NOT be specified in the <query> component.

### 1399 Step 6: Accessing the Identity Provider Return URL

1400 In step 6, the user agent accesses the identity provider's return URL location fulfilling the redirect  
1401 request.

### 1402 Step 7: Confirmation

1403 In step 7, the user agent is sent an HTTP response that confirms the requested action of identity  
1404 federation termination with the specific service provider.

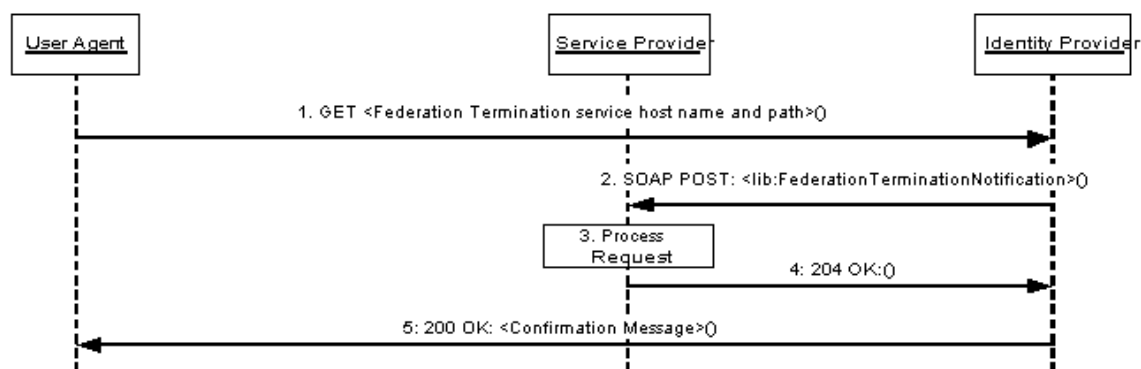
#### 1405 3.4.1.2 SOAP/HTTP-Based Profile

1406 The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP to communicate  
1407 federation termination notification messages from the identity provider to the service provider. See  
1408 Figure 9.

1409 The following URI-based identifier MUST be used when referencing this specific profile:

1410 URI: <http://projectliberty.org/profiles/fedterm-idp-soap>

1411 This URI identifier MUST be specified in the service provider metadata element  
1412 FederationTerminationNotificationProtocolProfile when the service provider intends  
1413 to indicate to the identity provider a preference for receiving federation termination notifications via  
1414 SOAP over HTTP.



1415

1416 **Figure 9: SOAP/HTTP-based profile for federation termination**

1417 This profile description assumes the following preconditions:

- 1418 • The Principal's identity at the service provider is federated with his/her identity at the  
1419 identity provider.
- 1420 • The Principal has requested to the identity provider that the federation be terminated.
- 1421 • The Principal has authenticated with the identity provider.

1422 **Step 1: Accessing the Federation Termination Service**

1423 In step 1, the user agent accesses the identity federation termination service URL at the identity  
1424 provider specifying the service provider for with whom identity federation termination should occur.  
1425 How the service provider is specified is implementation-dependent and, as such, is out of the scope  
1426 of this specification.

1427 **Step 2: Notification of Federation Termination**

1428 In step 2, the identity provider sends an asynchronous SOAP over HTTP notification message to the  
1429 service provider's SOAP endpoint. The SOAP message MUST contain exactly one  
1430 `<lib:FederationTerminationNotification>` element in the SOAP body and adhere to the  
1431 construction rules defined in [[LibertyProtSchema](#)].

1432 If a SOAP fault occurs, the identity provider SHOULD employ best effort to resolve the fault  
1433 condition and resend the federation termination notification message to the service provider.

1434 **Step 3: Processing the Notification**

1435 In step 3, the service provider MUST process the  
1436 `<lib:FederationTerminationNotification>` according to the rules defined in  
1437 [[LibertyProtSchema](#)].

1438 The service provider MAY remove any locally stored references to the name identifier it received  
1439 from the identity provider in the `<lib:FederationTerminationNotification>`.

1440 **Step 4: Responding to the Notification**

1441 In step 4, the service provider MUST respond to the  
1442 `<lib:FederationTerminationNotification>` with a HTTP 204 OK response.

1443 **Step 5: Confirmation**

1444 In step 5, the user agent is sent an HTTP response that confirms the requested action of identity  
1445 federation termination with the specific service provider.

1446 **3.4.2 Federation Termination Notification Initiated at Service Provider**

1447 The profiles in 3.4.2.1 and 3.4.2.2 are specific to identity federation termination notification when  
1448 initiated by a Principal at the service provider. Effectively, when using this profile, the service  
1449 provider is stating to the identity provider that the Principal has requested that the identity provider  
1450 no longer provide the Principal's identity information to the service provider and that service  
1451 provider will no longer ask the identity provider to do anything on the behalf of the Principal.

1452 It is RECOMMENDED that the service provider, after initiating or receiving a federation  
1453 termination notification, invalidate the local session for the Principal that was authenticated at the  
1454 identity provider with which federation has been terminated. If the Principal was locally  
1455 authenticated at the service provider, the service provider MAY continue to maintain a local session  
1456 for the Principal. If the Principal wants to engage in a single sign-on session with identity provider  
1457 again, the service provider MUST first federate with identity provider the given Principal.

### 1458 3.4.2.1 HTTP-Redirect-Based Profile

1459 The HTTP-redirect-based profile relies on the use of a HTTP 302 redirect to communicate a  
1460 federation termination notification message from the service provider to the identity provider. For a  
1461 discussion of the interactions and processing steps, refer to 3.4.1.1. When reviewing that profile,  
1462 interchange all references to service provider and identity provider in the interaction diagram and  
1463 processing steps.

1464 The following URI-based identifier **MUST** be used when referencing this specific profile:

1465 URI: `http://projectliberty.org/profiles/fedterm-sp-http`

1466 This URI identifier is really only meant for service provider consumption and as such is not needed  
1467 in any provider metadata.

### 1468 3.4.2.2 SOAP/HTTP-Based Profile

1469 The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP to communicate  
1470 federation termination notification messages from the service provider to the identity provider. For a  
1471 discussion of the interactions and processing steps, refer to 3.4.1.2. When reviewing that profile,  
1472 interchange all references to service provider and identity provider in the interaction diagram and  
1473 processing steps.

1474 The following URI-based identifier **MUST** be used when referencing this specific profile:

1475 URI: `http://projectliberty.org/profiles/fedterm-sp-soap`

1476 This URI identifier is really only meant for service provider consumption and as such is not needed  
1477 in any provider metadata.

## 1478 3.5 Single Logout Profiles

1479 The single logout profiles synchronize session logout functionality across all sessions that were  
1480 authenticated by a particular identity provider. The single logout can be initiated at either the identity  
1481 provider or the service provider. In either case, the identity provider will then communicate a logout  
1482 notification to each service provider with which it has established a session for the Principal. The  
1483 negotiation of which single logout profile the identity provider uses to communicate with each  
1484 service provider is based upon the `SingleLogoutProtocolProfile` provider metadata element  
1485 defined in [[LibertyProtSchema](#)].

1486 The available profiles are defined in 3.5.1 and 3.5.2, depending on whether the single logout is  
1487 initiated at the identity provider or service provider:

#### 1488 • Single Logout Initiated at Identity Provider

1489 • **HTTP-Based:** Relies on using either HTTP 302 redirects or HTTP GET requests to  
1490 communicate logout notifications from an identity provider to the service providers.

1491 • **SOAP/HTTP-Based:** Relies on asynchronous SOAP over HTTP messaging to  
1492 communicate logout notifications from an identity provider to the service providers.

#### 1493 • Single Logout Initiated at Service Provider

1494 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate a logout  
1495 notification with the identity provider.

1496 • **SOAP/HTTP-Based:** Relies on asynchronous SOAP over HTTP messaging to  
1497 communicate a logout notification from a service provider to an identity provider.

1498 The single logout profiles make use of the following metadata elements, as defined in  
1499 [[LibertyProtSchema](#)].

- 1500 • `SingleLogoutServiceURL` — The URL at the service provider or identity provider to  
1501 which single logout notifications are sent. It is described in these profiles as “single logout  
1502 service URL.”
- 1503 • `SingleLogoutServiceReturnURL` — The URL used by the service provider when  
1504 redirecting the user agent to the identity provider at the end of the single logout profile  
1505 process.
- 1506 • `SingleLogoutProtocolProfile` — Used by the identity provider to determine which  
1507 single logout notification profile **MUST** be used when communicating with the service  
1508 provider.
- 1509 • `SOAPEndpoint` — The SOAP endpoint location at the service provider or identity provider  
1510 to which Liberty SOAP messages are sent.

### 1511 **3.5.1 Single Logout Initiated at Identity Provider**

1512 The profiles in 3.5.1.1 through 3.5.1.2 are specific to single logout when initiated by a user agent at  
1513 the identity provider.

#### 1514 **3.5.1.1 HTTP-Based Profile**

1515 The HTTP-based profile defines two possible implementations that an identity provider may choose  
1516 to use. The first implementation relies on using HTTP 302 redirects, while the second used HTTP  
1517 GET requests. The choice of implementation is entirely dependent upon what type of user  
1518 experience the identity provider wants to provide.

1519 The following URI-based identifier **MUST** be used when referencing either implementation for this  
1520 specific profile:

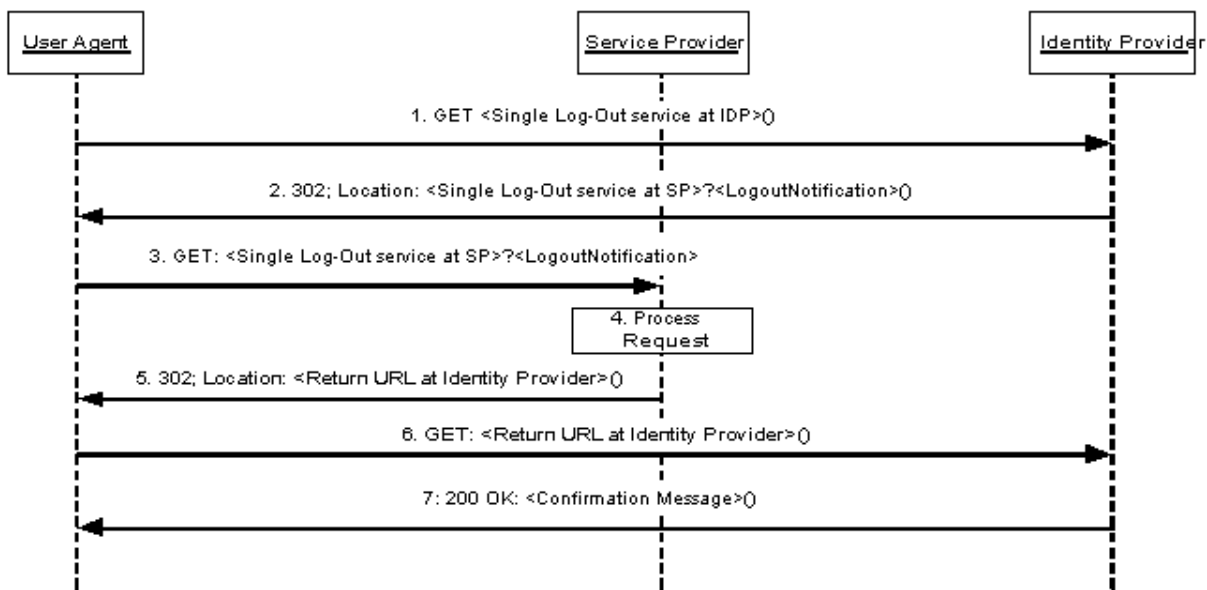
1521 URI: `http://projectliberty.org/profiles/slo-idp-http`

1522 This URI identifier **MUST** be specified in the service provider metadata element  
1523 `SingleLogoutProtocolProfile` when the service provider intends to indicate to the identity  
1524 provider a preference for receiving logout notifications via either a HTTP redirect or a HTTP GET.

##### 1525 **3.5.1.1.1 HTTP-Redirect Implementation**

1526 The HTTP-Redirect implementation uses HTTP 302 redirects to communicate a logout notification  
1527 to each service provider for which the identity provider has provided authentication assertions during  
1528 the Principal’s current session if the service provider indicated a preference to receive logout  
1529 notification via the HTTP based profile. See Figure 10.





1530

1531

**Figure 10: HTTP-Redirect implementation for single log out initiated at identity provider**

1532

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

1533

1534

1535

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

1536

1537

1538

1539

1540

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

1541

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.

1542

1543

1544

### Step 2: Redirecting to the Single Logout Service at the Service Provider

1545

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

1546

1547

1548

The redirections MUST adhere to the following rules:

1549

- The Location HTTP header MUST be set to the service provider’s single logout service URL.

1550

1551

- The service provider’s single logout service URL MUST specify https as the URL scheme; if another scheme is specified, the identity provider MUST NOT redirect to the service provider.

1552

1553

- 1554       • The Location HTTP header MUST include a <query> component containing the  
1555       <lib:LogoutNotification> protocol message as defined in [[LibertyProtSchema](#)] with  
1556       formatting as specified in 3.1.2.

1557       Note: Additionally, the URL-encoded <lib:LogoutNotification> message  
1558       MAY also include a parameter named RELAYSTATE with a value set to the URL  
1559       (and/or other information) to be used by the identity provider in the HTTP response  
1560       to the user agent at the completion of federation termination in step 7.

1561       The HTTP response MUST take the following form:

```
1562 <HTTP-Version> 302 <Reason Phrase>  
1563 <other headers>  
1564 Location : https://<Service Provider Single Log-Out service URL>?<query>  
1565 <other HTTP 1.0 or 1.1 components>
```

1566       where

1567       <Service Provider Single Log-Out service URL>

1568       This element provides the host name, port number, and path components of the single logout  
1569       service URL at the service provider.

1570       <query>= ...<URL-encoded LogoutNotification>...

1571       The <query> MUST contain a single logout notification request.

### 1572       **Step 3: Accessing the Service Provider Single Logout Service**

1573       In step 3, the user agent accesses the service provider's single logout service URL with the  
1574       <lib:LogoutNotification> information attached to the URL fulfilling the redirect request.

### 1575       **Step 4: Processing the Notification**

1576       In step 4, the service provider MUST process the <lib:LogoutNotification> according to the  
1577       rules defined in [[LibertyProtSchema](#)].

1578       The service provider MUST invalidate the session(s) of the Principal referred to in the name  
1579       identifier it received from the identity provider in the <lib:LogoutNotification>.

### 1580       **Step 5: Redirecting to the Identity Provider Return URL**

1581       In step 5, the service provider's single logout service responds and redirects the user agent back to  
1582       the identity provider using the return URL location obtained from the  
1583       SingleLogoutServiceReturnURL metadata element. If the URL-encoded  
1584       <lib:LogoutNotification> message received in step 3 contains a parameter named  
1585       RELAYSTATE, then the service provider MUST include a <query> component containing the same  
1586       RELAYSTATE parameter and its value in its response to the identity provider.

1587       No success or failure message should be conveyed in this HTTP redirect. The sole purpose of this  
1588       redirect is to return the user agent to the identity provider so that the single logout process may  
1589       continue in the same fashion with other service providers.

1590       The HTTP response MUST take the following form:

```
1591 <HTTP-Version> 302 <Reason Phrase>  
1592 <other headers>  
1593 Location : https://<Identity Provider Return URL>?<query>  
1594 <other HTTP 1.0 or 1.1 components>
```

1595       where

1596       <Identity Provider Return URL>

1597 This element provides the host name, port number, and path components of the return URL at the  
1598 identity provider.

1599 <query>= ...RELAYSTATE=<...>

1600 The <query> component MUST contain the identical RELAYSTATE parameter and its value that  
1601 was received in the URL-encoded logout notification message obtained in step 3. If no  
1602 RELAYSTATE parameter was provided in the step 3 message, then a RELAYSTATE parameter  
1603 MUST NOT be specified in the <query> component.

#### 1604 Step 6: Accessing the Identity Provider Return URL

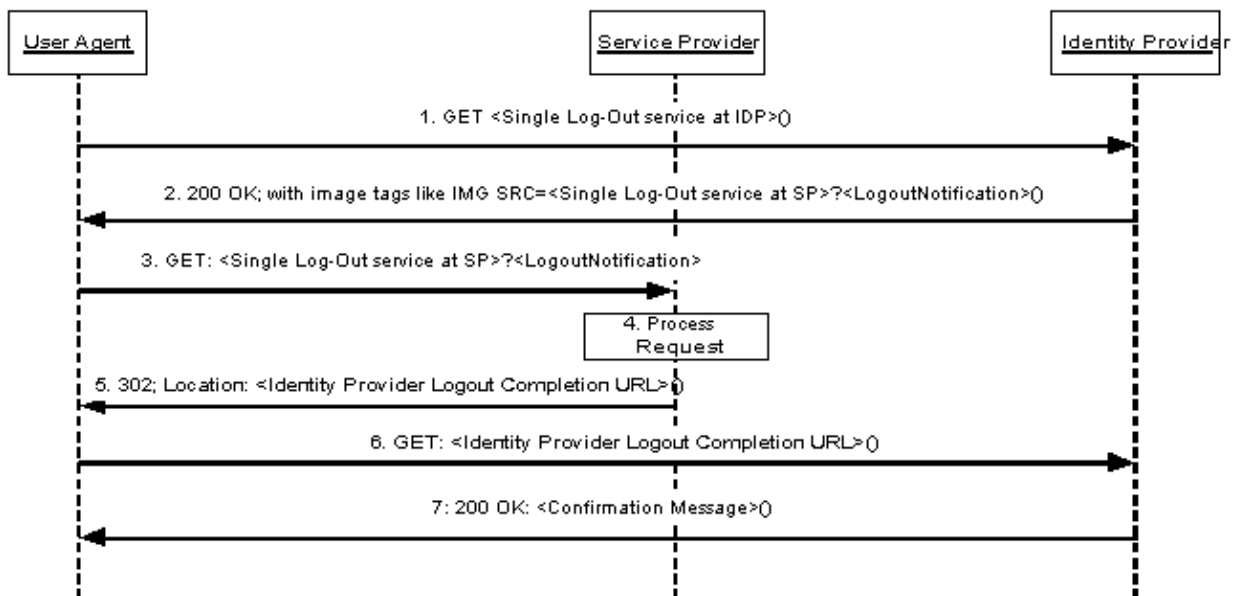
1605 In step 6, the user agent accesses the identity provider's return URL fulfilling the redirect  
1606 request.

#### 1607 Step 7: Confirmation

1608 In step 7, the user agent is sent an HTTP response that confirms the requested action of single logout  
1609 has completed.

### 1610 3.5.1.1.2 HTTP-GET Implementation

1611 The HTTP-GET implementation uses HTTP GET requests to communicate logout notifications to  
1612 each service provider for which the identity provider has provided authentication during the  
1613 Principal's current session if the service provider indicated a preference to receive logout  
1614 notifications via the HTTP based profile. See Figure 11.



1615

1616 **Figure 11: HTTP-GET implementation for single logout initiated at identity provider**

1617 Note: Steps 3 through 7 may be an iterative process for each service provider that has been issued  
1618 authentication assertions during the Principal's current session and has indicated a preference to  
1619 receive logout notification via the HTTP based profile.

1620 **Step 1: Accessing the Single Logout Service at the Identity Provider**

1621 In step 1, the user agent accesses the single logout service URL at the identity provider indicating  
1622 that all service providers for which this identity provider has provided authentication assertions  
1623 during the Principal's current session must be notified of session termination.

1624 **Step 2: HTML Page Returned to User Agent with Image Tags**

1625 In step 2, the identity provider's single logout service responds with an HTML page that includes  
1626 image tags referencing the logout service URL for each of the service providers for which the  
1627 identity provider has provided an authentication assertion during the Principal's current session. The  
1628 list of image tags MUST be sent in a standard HTTP 200 response to the user agent.

1629 The image tag loads on the HTML page MUST adhere to the following rules:

- 1630 • The SRC attribute MUST be set to the specific service provider's single logout service URL.
- 1631 • The service provider's single logout service URL MUST specify https as the URL scheme.
- 1632 • The service provider's single logout service URL MUST include a <query> component  
1633 containing the <lib:LogoutNotification> protocol message as defined in  
1634 [[LibertyProtSchema](#)] with formatting as specified in 3.1.2.

1635 **Step 3: Accessing the Service Provider Single Logout Service**

1636 In step 3, the user agent, as a result of each image load, accesses the service provider's single logout  
1637 service URL with <lib:LogoutNotification> information attached to the URL. This step may  
1638 occur multiple times if the HTTP response includes multiple image tag statements (one for each  
1639 service provider that has been issued authentication assertions during the Principal's current session).

1640 **Step 4: Processing the Notification**

1641 In step 4, the service provider MUST process the <lib:LogoutNotification> according to the  
1642 rules defined in [[LibertyProtSchema](#)].

1643 The service provider MUST invalidate the session of the Principal referred to in the name identifier it  
1644 received from the identity provider in the <lib:LogoutNotification>.

1645 **Step 5: Redirecting to the Identity Provider Logout Completion URL**

1646 In step 5, the service provider's single logout service responds and redirects the image load back to  
1647 the identity provider's logout completion URL. This location will typically point to an image that  
1648 will be loaded by the user agent to indicate that the logout is complete (for example, a checkmark).

1649 The logout completion URL is obtained from the SingleLogoutServiceReturnURL metadata  
1650 element.

1651 The HTTP response MUST take the following form:

```
1652 <HTTP-Version> 302 <Reason Phrase>  
1653 <other headers>  
1654 Location : https://<Identity Provider Logout Completion URL>  
1655 <other HTTP 1.0 or 1.1 components>
```

1656 where

1657 <Identity Provider Logout Completion URL>

1658 This element provides the host name, port number, and path components of the identity provider

1659       logout completion URL at the identity provider.

1660   **Step 6: Accessing the Identity Provider Logout Completion URL**

1661   In step 6, the user agent accesses the identity provider’s logout completion URL fulfilling the  
1662   redirect request.

1663   **Step 7: Confirmation**

1664   In step 7, the user agent is sent an HTTP response that confirms the requested action of single logout  
1665   has completed.

1666   Note: One method for seamlessly returning the user agent back to the identity provider is for the  
1667   HTML page generated in step 2 to include a script that runs when the page is completely loaded (all  
1668   logouts completed) that will initiate the redirect to the identity provider.

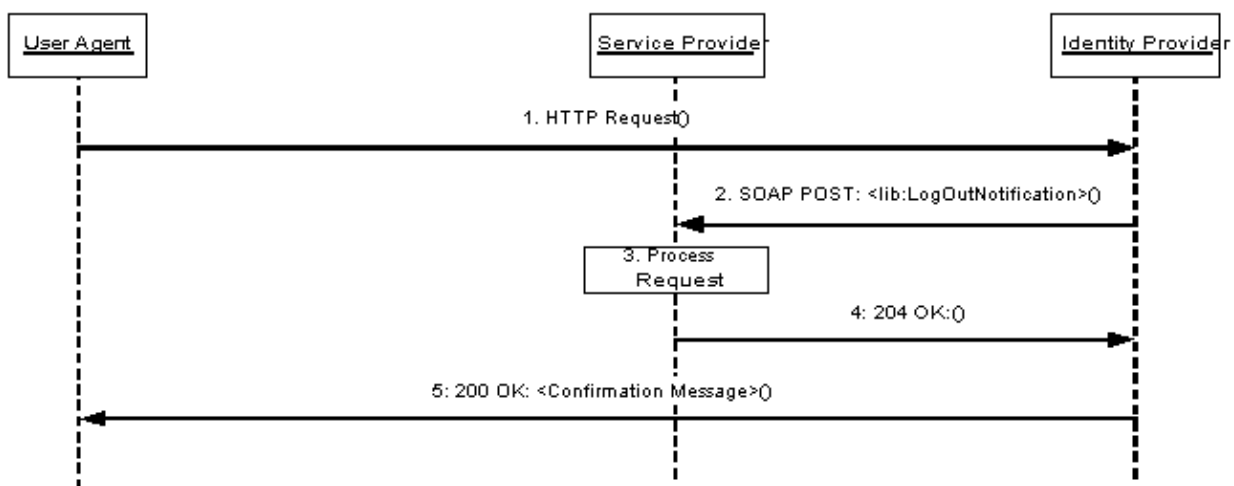
1669   **3.5.1.2   SOAP/HTTP-Based Profile**

1670   The SOAP/HTTP-based profile uses asynchronous SOAP over HTTP messaging to communicate a  
1671   logout notification to each service provider for which the identity provider has provided  
1672   authentication assertions during the Principal’s current session if the service provider indicated a  
1673   preference to receive logout notification via the SOAP/HTTP-based profile. See Figure 12.

1674   The following URI-based identifier **MUST** be used when referencing this specific profile:

1675       URI: `http://projectliberty.org/profiles/slo-idp-soap`

1676   This URI identifier **MUST** be specified in the service provider metadata element  
1677   `SingleLogoutProtocolProfile` when the service provider intends to indicate to the identity  
1678   provider a preference for receiving logout notifications via SOAP over HTTP.



1679  
1680   **Figure 12: SOAP/HTTP-based profile for single logout initiated at identity provider**

1681   Note: Steps 2 through 4 may be an iterative process for each service provider that has been issued  
1682   authentication assertions during the Principal’s current session and has indicated a preference to  
1683   receive logout notifications via the SOAP/HTTP asynchronous message profile.

1684 **Step 1: Accessing the Single Logout Service**

1685 In step 1, the user agent accesses the single logout service URL at the identity provider via an HTTP  
1686 request.

1687 **Step 2: Notification of Logout**

1688 In step 2, the identity provider sends an asynchronous SOAP over HTTP notification message to the  
1689 SOAP endpoint of each service provider for which it provided authentication assertions during the  
1690 Principal's current session. The SOAP message MUST contain exactly one  
1691 `<lib:logoutNotification>` element in the SOAP body and adhere to the construction rules  
1692 defined in [[LibertyProtSchema](#)].

1693 If a SOAP fault occurs, the identity provider SHOULD employ best effort to resolve the fault  
1694 condition and resend the single logout notification message to the service provider.

1695 **Step 3: Processing the Notification**

1696 In step 3, the service provider MUST process the `<lib:LogoutNotification>` according to the  
1697 rules defined in [[LibertyProtSchema](#)].

1698 The service provider MUST invalidate the session for the Principal specified by the name identifier  
1699 provided by the identity provider in the `<lib:LogoutNotification>`.

1700 **Step 4: Responding to the Notification**

1701 In step 4, the service provider MUST respond to the `<lib:LogoutNotification>` with a HTTP  
1702 204 OK response.

1703 **Step 5: Confirmation**

1704 In step 5, the user agent is sent an HTTP response that confirms the requested action of single logout  
1705 has completed.

1706 **3.5.2 Single Logout Initiated at Service Provider**

1707 The profiles in 3.5.2.1 and 3.5.2.2 are specific to when the Principal initiates the single logout  
1708 notification process at the service provider.

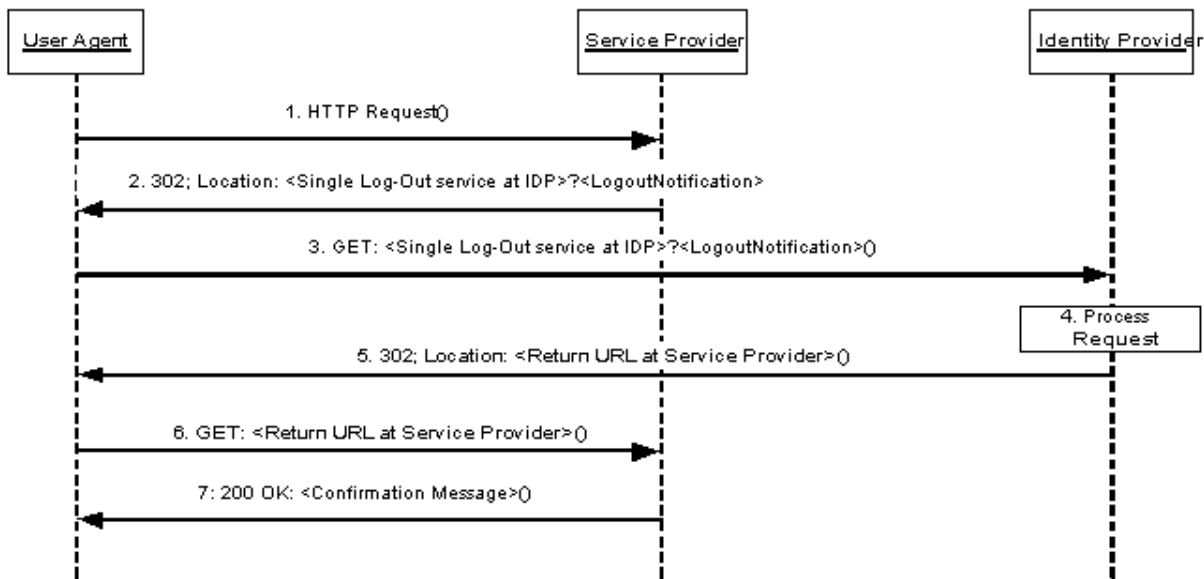
1709 **3.5.2.1 HTTP-Redirect-Based Profile**

1710 The HTTP-redirect-based profile relies on using a HTTP 302 redirect to communicate a logout  
1711 notification with the identity provider. The identity provider will then communicate a logout  
1712 notification to each service provider with which it has established a session for the Principal using  
1713 the service provider's preferred profile for logout notification from the identity provider (see 3.5.1).  
1714 See Figure 13.

1715 The following URI-based identifier MUST be used when referencing this specific profile:

1716 URI: `http://projectliberty.org/profiles/slo-sp-http`

1717 This URI identifier is really only meant for service provider consumption and as such is not needed  
1718 in any provider metadata.



1719

1720 **Figure 13: HTTP-redirect-based profile for single logout initiated at service provider**

1721 Note: Step 4 may involve an iterative process by the identity provider to implement the preferred  
 1722 profile for logout notification for each service provider that has been issued authentication assertions  
 1723 during the Principal's current session.

1724 **Step 1: Accessing the Single Logout Service at the Service Provider**

1725 In step 1, the user agent accesses the single logout service URL at the service provider indicating that  
 1726 session logout is desired at the associated identity provider and all service providers for which this  
 1727 identity provider has provided authentication assertions during the Principal's current session. If a  
 1728 current session exists for the Principal at the service provider, it is RECOMMENDED that the  
 1729 service provider terminate that session prior to step 2.

1730 **Step 2: Redirecting to the Single Logout Service at the Identity Provider**

1731 In step 2, the service provider's single logout service responds and redirects the user agent to the  
 1732 single logout service URL at the identity provider.

1733 The redirection MUST adhere to the following rules:

- 1734 • The Location HTTP header MUST be set to the identity provider's single logout service  
 1735 URL.
- 1736 • The identity provider's single logout service URL MUST specify `https` as the URL  
 1737 scheme; if another scheme is specified, the service provider MUST NOT redirect to the  
 1738 identity provider.
- 1739 • The Location HTTP header MUST include a `<query>` component containing the  
 1740 `<lib:LogoutNotification>` protocol message as defined in [[LibertyProtSchema](#)] with  
 1741 formatting as specified in 3.1.2.

1742 Note: Additionally, the URL-encoded `<lib:LogoutNotification>` message  
 1743 MAY also include a parameter named `RELAYSTATE` with a value set to the URL



1744 (and/or other information) to be used by the identity provider in the HTTP response  
1745 to the user agent at the completion of federation termination in step 7.

1746 The HTTP response MUST take the following form:

```
1747 <HTTP-Version> 302 <Reason Phrase>  
1748 <other headers>  
1749 Location : https://<Identity Provider single log-out service URL>?<query>  
1750 <other HTTP 1.0 or 1.1 components>
```

1751 where

1752 <Identity Provider single log-out service URL>

1753 This element provides the host name, port number, and path components of the single logout  
1754 service URL at the identity provider.

1755 <query>= ...<URL-encoded LogoutNotification>...

1756 The <query> MUST contain a single logout notification request.

### 1757 Step 3: Accessing the Identity Provider Single Logout Service

1758 In step 3, the user agent accesses the identity provider's single logout service URL with the  
1759 <lib:LogoutNotification> information attached to the URL fulfilling the redirect request.

### 1760 Step 4: Processing the Request

1761 In step 4, the identity provider MUST process the <lib:LogoutNotification> according to the  
1762 rules defined in [[LibertyProtSchema](#)].

1763 Each service provider for which the identity provider has provided authentication assertions during  
1764 the Principal's current session MUST be notified via the service provider's preferred profile for  
1765 logout notification from the identity provider (see 3.5.1).

1766 The identity provider's current session with the Principal MUST be terminated, and no more  
1767 authentication assertions for the Principal are to be given to service providers.

### 1768 Step 5: Redirecting to the Service Provider Return URL

1769 In step 5, the identity provider's single logout service responds and redirects the user agent back to  
1770 service provider using the return URL location obtained from the  
1771 SingleLogoutServiceReturnURL metadata element. If the URL-encoded  
1772 <lib:LogoutNotification> message received in step 3 contains a parameter named  
1773 RELAYSTATE, then the identity provider MUST include a <query> component containing the same  
1774 RELAYSTATE parameter and its value in its response to the service provider.

1775 No success or failure message should be conveyed in this HTTP redirect. The sole purpose of this  
1776 redirect is to return the user agent to the service provider.

1777 The HTTP response MUST take the following form:

```
1778 <HTTP-Version> 302 <Reason Phrase>  
1779 <other headers>  
1780 Location : https://<Service Provider Return URL>?<query>  
1781 <other HTTP 1.0 or 1.1 components>
```

1782 where

1783 <Service Provider Return URL>

1784 This element provides the host name, port number, and path components of the return URL  
1785 location at the service provider.

1786 <query>= ...RELAYSTATE=<...>

1787 The <query> component MUST contain the identical RELAYSTATE parameter and its value that  
1788 was received in the URL-encoded logout notification message obtained in step 3. If no  
1789 RELAYSTATE parameter was provided in the step 3 message, then a RELAYSTATE parameter  
1790 MUST NOT be specified in the <query> component.

### 1791 Step 6: Accessing the Service Provider Return URL

1792 In step 6, the user agent accesses the service provider's return URL location fulfilling the redirect  
1793 request.

### 1794 Step 7: Confirmation

1795 In step 7, the user agent is sent an HTTP response that confirms the requested action of single logout  
1796 has completed.

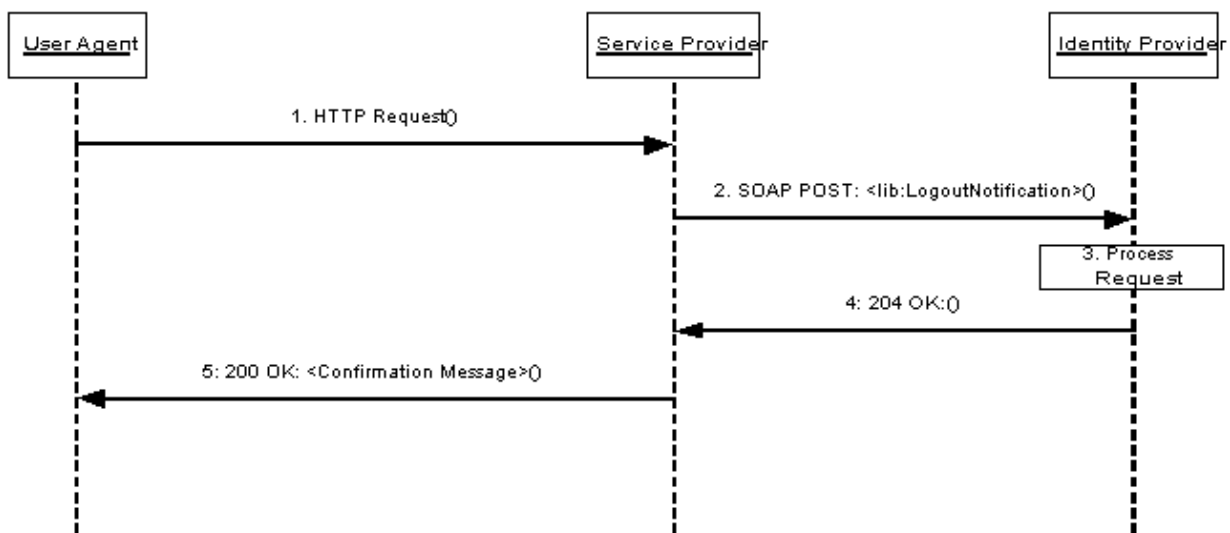
### 1797 3.5.2.2 SOAP/HTTP-Based Profile

1798 The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP messages to  
1799 communicate a logout notification with the identity provider. The identity provider will then  
1800 communicate a logout notification to each service provider it has established a session with for the  
1801 Principal via the service provider's preferred profile for logout notification from the identity provider  
1802 (see 3.5.1). See Figure 14.

1803 The following URI-based identifier MUST be used when referencing this specific profile:

1804 URI: `http://projectliberty.org/profiles/slo-sp-soap`

1805 This URI identifier is really only meant for service provider consumption and as such is not needed  
1806 in any provider metadata.



1807

1808 **Figure 14: SOAP/HTTP-based profile for single logout initiated at service provider**

1809 Note: Step 3 may involve an iterative process by the identity provider to implement the preferred  
1810 profile for logout notification for each service provider that has been issued authentication assertions  
1811 during the Principal's current session.

1812 **Step 1: Accessing Single Logout Service**

1813 In step 1, the user agent accesses the single logout service URL at the service provider via an HTTP  
1814 request.

1815 **Step 2: Notification of Logout**

1816 In step 2, the service provider sends an asynchronous SOAP over HTTP notification message to the  
1817 identity provider's SOAP endpoint. The SOAP message MUST contain exactly one  
1818 `<lib:LogoutNotification>` element in the SOAP body and adhere to the construction rules as  
1819 defined in [[LibertyProtSchema](#)].

1820 If a SOAP fault occurs, the service provider SHOULD employ best effort to resolve the fault  
1821 condition and resend the single logout notification message to the identity provider.

1822 **Step 3: Processing the Notification**

1823 In step 3, the identity provider MUST process the `<lib:LogoutNotification>` according to the  
1824 rules defined in [[LibertyProtSchema](#)].

1825 Each service provider for which the identity provider has provided authentication assertions during  
1826 the Principal's current session MUST be notified via the service provider's preferred profile for  
1827 logout notification from the identity provider. If any of the active service provider's preferred single  
1828 logout profile is HTTP-based, then the identity provider MUST respond to the sender of the  
1829 `<lib:LogoutNotification>` message in step 2 with a SOAP fault. The SOAP fault MUST  
1830 adhere to the following rules;

- 1831 • The `faultcode` MUST specify "Server."
- 1832 • The `faultstring` must specify "Cannot execute Single Log Out using web service."

1833 This SOAP fault condition will indicate to the service provider that it MUST redirect the user agent  
1834 to the identity provider to achieve single logout.

1835 The identity provider's current session with the Principal MUST be terminated, and no more  
1836 authentication assertions for the Principal are to be given to service providers.

1837 **Step 4: Responding to the Notification**

1838 In step 4, the identity provider MUST respond to the `<lib:LogoutNotification>` with a HTTP  
1839 204 OK response, unless a SOAP error was sent as specified in step 3.

1840 **Step 5: Confirmation**

1841 In step 5, the user agent is sent an HTTP response that confirms the requested action of single logout  
1842 was completed.

1843 **3.6 Identity Provider Introduction**

1844 This section defines the profiles by which a service provider discovers which identity providers a  
1845 Principal is using. In identity federation networks having more than one identity provider, service  
1846 providers need a means to discover which identity providers a Principal uses. The introduction  
1847 profile relies on a cookie that is written in a domain that is common between identity providers and  
1848 service providers in an identity federation network. The domain that the identity federation network

1849 predetermines for a deployment is known as the *common domain* in this specification, and the cookie  
1850 containing the list of identity providers is known as the *common domain cookie*.

1851 Implementation of this profile is OPTIONAL. Whether identity providers and service providers  
1852 implement this profile is a policy and deployment issue outside the scope of this specification. Also,  
1853 which entities host web servers in the common domain is a deployment issue and is outside the scope  
1854 of this specification.

### 1855 3.6.1 Common Domain Cookie

1856 The name of the cookie MUST be `_liberty_idp`. The format of the cookie content MUST be a list  
1857 of base64-encoded (see [\[RFC2045\]](#)) identity provider succinct IDs separated by a single white space  
1858 character. The identity provider IDs MUST adhere to the creation rules as defined in 3.2.2.2. The  
1859 identity provider ID is a metadata element, as described in 3.1.3 and defined in [\[LibertyProtSchema\]](#).

1860 The cookie MUST be set with no `Path` prefix or a `Path` prefix of `"/`. The `Domain` MUST be set to  
1861 `".[common-domain]"` where `[common-domain]` is the *common domain* established within the  
1862 identity federation network for use with the introduction protocol. The cookie MUST be marked as  
1863 `Secure`.

1864 The cookies MAY be either session or persistent (see [\[RFC2109\]](#)), the implementation of which is a  
1865 policy and deployment issue of the identity federation network.

### 1866 3.6.2 Setting the Common Domain Cookie

1867 After the identity provider authenticates a Principal, it MAY set the common domain cookie. The  
1868 means by which the identity provider sets the cookie are implementation-specific so long as the  
1869 cookie is successfully set with the parameters given above. One possible implementation strategy  
1870 follows and should be considered non-normative. The identity provider may:

- 1871 • Have previously established a DNS and IP alias for itself in the common domain
- 1872 • Redirect the user agent to itself using the DNS alias using a URL specifying "https" as  
1873 the URL scheme. The structure of the URL is private to the implementation and may  
1874 include session information needed to identify the user-agent.
- 1875 • Set the cookie on the redirected user agent using the parameters specified above
- 1876 • Redirect the user agent back to itself, or, if appropriate, to the service provider.

### 1877 3.6.3 Obtaining the Common Domain Cookie

1878 When a service provider needs to discover which identity providers the Principal uses, it invokes a  
1879 protocol exchange designed to present the common domain cookie to the service provider after it is  
1880 read by an HTTP server in the common domain.

1881 If the HTTP server in the common domain is operated by the service provider, the service provider  
1882 MAY redirect the user agent to an identity provider's intersite transfer service for an optimized  
1883 single sign-on process.

1884 The specific means by which the service provider reads the cookie are implementation-specific so  
1885 long as it is able to cause the user agent to present cookies that have been set with the parameters  
1886 given in section 3.6.1. One possible implementation strategy is described as follows and should be  
1887 considered non-normative. Additionally, it may be sub-optimal for some applications.

- 1888 • Have previously established a DNS and IP alias for itself in the common domain

- 1889
- 1890
- 1891
- 1892
- 1893
- 1894
- Redirect the user agent to itself using the DNS alias using a URL specifying "https" as the URL scheme. The structure of the URL is private to the implementation and should encode session state information needed to identify the user-agent, since it will not present any session cookies that were set using the service provider's original DNS name. This redirection could be performed via an HTTP 302 response or via an image tag in an HTML page sent to the user agent.
- 1895
- Read the cookie sent by the user agent, if any. Using the session state information encoded in the request URL, store the information retrieved from the cookie.
- 1896
- Use the session state information to determine what resource the user agent was attempting to access when it was redirected, if necessary. Send the user agent an HTTP 302 response redirecting it to the original resource.
- 1897
- 1898
- 1899
- 1900

## 1901 4 Security Considerations

### 1902 4.1 Introduction

1903 This section describes security considerations associated with Liberty protocols for identity  
1904 federation, single sign-on, federation termination, and single logout.

1905 Liberty protocols, schemas, bindings, and profiles inherit and heavily utilize the SAML protocols.  
1906 Therefore, the security considerations published along with the SAML specification have direct  
1907 relevance (see [[SAMLCore](#)], [[SAMLBind](#)], and [[SAMLSec](#)]). Throughout this section if, for any  
1908 reason, a specific consideration or countermeasure does not apply or differs, notice of this fact is  
1909 made; and a description of alternatives is supplied, where possible.

### 1910 4.2 General Requirements

#### 1911 4.2.1 Security of SSL and TLS

1912 SSL and TLS utilize a suite of possible cipher suites. The security of the SSL or TLS session  
1913 depends on the chosen cipher suite. An entity (that is, a user agent, service provider, or identity  
1914 provider) that terminates an SSL or TLS connection needs to offer (or accept) suitable cipher suites  
1915 during the handshake. The following list of TLS 1.0 cipher suites (or their SSL 3.0 equivalent) is  
1916 recommended.

- 1917
- TLS\_RSA\_WITH\_RC4\_128\_SHA
- 1918
- TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- 1919
- TLS\_DHE\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA

1920 The above list is by no means exhaustive. The recommended cipher suites are among the most  
1921 commonly used. Note: New cipher suites are added as they are standardized and should be  
1922 considered for inclusion if they have sufficiently strong security properties. For example, it is  
1923 anticipated that the AES-based cipher suites being standardized in the IETF will be widely adopted  
1924 and deployed.

1925 **4.2.2 Security Implementation**

1926 The suitable implementation of security protocols is necessary to maintain the security of a system,  
1927 including

- 1928 • Secure random or pseudo-random number generator
- 1929 • Secure storage

1930 **4.3 Classification of Threats**

1931 **4.3.1 Threat Model**

1932 For an analysis of threat classifications, an Internet threat model has been used. In other words, the  
1933 threat model assumes that intermediary and end-systems participating in Liberty protocol exchanges  
1934 have not been compromised. However, where possible, the consequences and containment properties  
1935 of a compromised system entity are described and countermeasures are suggested to bolster the  
1936 security posture so that the exposure from a security breach is minimized.

1937 Given the nature of the Internet, the assumption is made that deployment is across the global Internet  
1938 and, therefore, crosses multiple administrative boundaries. Thus, an assumption is also made that the  
1939 adversary has the capacity to engage in both passive and active attacks (see 4.3.3).

1940 **4.3.2 Rogue and Spurious Entities**

1941 Attackers may be classified based on their capabilities and the roles that they play in launching  
1942 attacks on a Liberty system as follows:

- 1943 • **Rogue Entities:** Entities that misuse their privileges. The rogue actors may be Principals,  
1944 user agents, service providers, or identity providers. A rogue Principal is a legitimate  
1945 participant who attempts to escalate its privileges or masquerade as another system Principal.  
1946 A rogue user agent may, for instance, misuse the relationships between its associated  
1947 Principals and an identity provider to launch certain attacks. Similarly, a rogue service  
1948 provider may be able to exploit the relationship that it has either with a Principal or with an  
1949 identity provider to launch certain attacks.
- 1950 • **Spurious Entities:** Entities that masquerade as a legitimate entity or are completely  
1951 unknown to the system. The spurious actors include Principals, user agents (i.e., user agents  
1952 without associated legitimate Liberty Principals), service providers, or identity providers. A  
1953 spurious service provider may, for instance, pretend to be a service provider that has a  
1954 legitimate relationship with an identity provider. Similarly, a spurious Principal may be one  
1955 that pretends to be a legitimate Principal that has a relationship with either a service provider  
1956 or an identity provider.

1957 **4.3.3 Active and Passive Attackers**

1958 Both rogue and spurious entities may be able to launch active or passive attacks on the system. A  
1959 passive attack is one where the attacker does not inject any traffic or modify any traffic in any way.  
1960 Such an attacker usually passively monitors the traffic flow, and the information that is obtained in  
1961 that flow may be used at a later time. An active attacker, on the other hand, is capable of modifying  
1962 existing traffic as well as injecting new traffic into the system.



1963 **4.3.4 Scenarios**

1964 The following scenarios describe possible attacks:

- 1965 • **Collusion: The secret cooperation between two or more Liberty entities to launch an**  
1966 **attack, for example,**
- 1967 • Collusion between Principal and service provider
  - 1968 • Collusion between Principal and identity provider
  - 1969 • Collusion between identity provider and service provider
  - 1970 • Collusion among two or more Principals
  - 1971 • Collusion between two or more service providers
  - 1972 • Collusion between two or more identity providers
- 1973 • **Denial-of-Service Attacks:** The prevention of authorized access to a system resource or the  
1974 delaying of system operations and functions.
- 1975 • **Man-in-the-Middle Attacks:** A form of active wiretapping attack in which the attacker  
1976 intercepts and selectively modifies communicated data to masquerade as one or more of the  
1977 entities involved in a communication association.
- 1978 • **Replay Attacks:** An attack in which a valid data transmission is maliciously or fraudulently  
1979 repeated, either by the originator or by an adversary who intercepts the data and retransmits  
1980 it, possibly as part of a masquerade attack.
- 1981 • **Session Hijacking:** A form of active wiretapping in which the attacker seizes control of a  
1982 previously established communication association.

1983 **4.4 Threat Scenarios and Countermeasures**

1984 In this section, threats that may apply to all the Liberty profiles are considered first. Then threats that  
1985 are specific to individual profiles are considered. In each discussion the threat is described as well as  
1986 what countermeasures exist in the profile or what additional countermeasures may be implemented  
1987 to mitigate the threat.

1988 **4.4.1 Common Threats for All Profiles**

1989 **Threat:** Request messages sent in cleartext

1990 **Description:** Most profile protocol exchanges do not mandate that all exchanges commence  
1991 over a secure communication channel. This lack of transport security potentially exposes  
1992 requests and responses to both passive and active attacks.

1993 One obvious manifestation is when the initial contact is not over a secure transport and the  
1994 Liberty profile begins to exchange messages by carrying the request message back to the user  
1995 agent in the location header of a redirect.

1996 Another such manifestation could be a request or response message which carries a URI that  
1997 may be resolved on a subsequent exchange, for instance `lib:AuthnContextClassRef`. If  
1998 this URI were to specify a less or insecure transport, then the exchange may be vulnerable to  
1999 the types of attacks described above.



2000           **Countermeasure:** Ensure that points of entry to Liberty protocol exchanges utilize the  
2001           https URL <scheme> and that all interactions for that profile consistently exchange  
2002           messages over https.

2003   **Threat:** Malicious redirects into identity or service provider targets

2004           **Description:** A spurious entity could issue a redirect to a user agent so that the user agent  
2005           would access a resource that disrupts single sign-on. For example, an attacker could redirect  
2006           the user agent to a logout resource of a service provider causing the Principal to be logged out  
2007           of all existing authentication sessions.

2008           **Countermeasure:** Access to resources that produce side effects could be specified with a  
2009           transient qualifier that must correspond to the current authentication session. Alternatively, a  
2010           confirmation dialog could be interposed that relies on a transient qualifier with similar  
2011           semantics.

2012   **Threat:** Relay state tampering or fabrication

2013           **Description:** Some of the messages may carry a <lib:RelayState> element, which is  
2014           recommended to be integrity-protected by the producer and optionally confidentiality-  
2015           protected. If these practices are not followed, an adversary could trigger unwanted side  
2016           effects. In addition, by not confidentiality-protecting the value of this element, a legitimate  
2017           system entity could inadvertently expose information to the identity provider or a passive  
2018           attacker.

2019           **Countermeasure:** Follow the recommended practice of confidentiality- and integrity-  
2020           protecting the <lib:RelayState> data. Note: Because the value of this element is both  
2021           produced and consumed by the same system entity, symmetric cryptographic primitives could  
2022           be utilized.

## 2023   4.4.2     Single Sign-On and Federation

### 2024   4.4.2.1   Common Interactions for All Single Sign-On and Federation Profiles

2025   **Threat:** <lib:AuthnRequest> sent over insecure channel

2026           **Description:** It is recommended that the initial exchange to access the intersite transfer  
2027           service be conducted over a TLS-secured transport. Not following this recommendation can  
2028           expose the exchange to both passive and active attacks.

2029           **Countermeasure:** Deploy the intersite transfer service under an https scheme.

2030   **Threat:** Unsigned <lib:AuthnRequest> message

2031           **Description:** The signature element of an <lib:AuthnRequest> is optional and thus the  
2032           absence of the signature could pose a threat to the identity provider or even the targeted  
2033           service provider. For example, a spurious system entity could generate an unsigned  
2034           <lib:AuthnRequest> and redirect the user agent to the identity provider. The identity  
2035           provider must then consume resources.

2036           **Countermeasure:** Sign the <lib:AuthnRequest>. The IDP can also verify the identity of  
2037           the Principal in the absence of a signed request.

2038   **Threat:** Replay of an authentication assertion

2039           **Description:** After obtaining a valid assertion from an identity provider, either legitimately or  
2040           surreptitiously, the entity replays the assertion to the Service at a later time. A digital

2041 signature must cover the entire assertion, thus elements within the assertion cannot be  
2042 corrupted without detection during the mandatory verification step. However, it is possible to  
2043 fabricate an `<lib:AuthnResponse>` with the valid assertion.

2044 **Countermeasure:** The issuer should sign `<lib:AuthnResponse>` messages. Signing binds  
2045 the `<samlp:IssueInstant>` of the response message to the assertion it contains. This  
2046 binding accords the relying party the opportunity to temporally judge the response.  
2047 Additionally, a valid signature over the response binds the `<samlp:InResponseTo>`  
2048 element to the corresponding `<lib:AuthnRequest>`. (Specifying a short period that the  
2049 authentication assertion can be relied upon will minimize, but not mitigate this threat.  
2050 Binding the `<lib:AssertionId>` to the request/`<samlp:InResponseTo>` element may  
2051 also be handy.)

2052 **Threat:** Fabricated `<lib:AuthnResponse>` denial of service

2053 **Description:** An attacker captures the `<samlp:RequestID>` sent in an  
2054 `<lib:AuthnRequest>` message by a service provider to an identity provider, and sends  
2055 several spurious `<lib:AuthnResponse>` messages to the service provider with the same  
2056 `<samlp:InResponseTo>`. Because the `<samlp:InResponseTo>` matches a  
2057 `<samlp:RequestID>` that the service provider had used, the service provider goes through  
2058 the process of validating the signature in the message. Thus, it is subject to a denial of service  
2059 attack.

2060 **Countermeasure:** A secure communication channel should be established before  
2061 transferring requests and responses.

2062 **Threat:** Collusion between two Principals

2063 **Description:** After getting an artifact or `<lib:AuthnResponse>` in step 6 (see 3.2), a  
2064 legitimate Principal A could pass this artifact or `<lib:AuthnResponse>` on to another  
2065 Principal, B. Principal B is now able to use the artifact or `<lib:AuthnResponse>`, while  
2066 the actual authentication happened via Principal A.

2067 **Countermeasure:** Implementations where this threat is a concern MUST use the  
2068 `<saml:AuthenticationLocality>` in the authentication statement. The IP address that  
2069 Principal B uses would be different from the IP address within the  
2070 `<saml:AuthenticationLocality>`. This countermeasure may not suffice when the user  
2071 agent is behind a firewall or proxy server. IP spoofing may also circumvent this  
2072 countermeasure.

2073 **Threat:** Stolen artifact and subsequent Principal impersonation

2074 **Description:** See Section 4.1.1.9.1 in [[SAMLBind](#)]

2075 **Countermeasure:** Identity providers MUST enforce a policy of one-time retrieval of the  
2076 assertion corresponding to an artifact so that a stolen artifact can be used only once.  
2077 Implementations where this threat is a concern MUST use the  
2078 `<saml:AuthenticationLocality>` in the authentication statement. The IP address of a  
2079 spurious user agent that attempts to use the stolen artifact would be different from IP address  
2080 within the `<saml:AuthenticationLocality>`. The service provider may then be able to  
2081 detect that the IP addresses differ. This countermeasure may not suffice when the user agent  
2082 is behind a firewall or proxy server. IP address spoofing may also circumvent this  
2083 countermeasure.

2084 **Threat:** Stolen assertion and subsequent Principal impersonation

2085           **Description:** See Section 4.1.1.9.1 in [[SAMLBind](#)]

2086           **Countermeasure:** Refer to the previous threat for requirements.

2087 **Threat:** Rogue service provider uses artifact or assertion to impersonate Principal at a different  
2088 service provider

2089           **Description:** Because the `<lib:AuthnResponse>` contains the `<lib:ProviderID>`, this  
2090 threat is not possible.

2091           **Countermeasure:** None

2092 **Threat:** Rogue identity provider impersonates Principal at a service provider

2093           **Description:** Because the Principal trusts the identity provider, it is assumed that the identity  
2094 provider does not misuse the Principal's trust.

2095           **Countermeasure:** None

2096 **Threat:** Rogue user attempts to impersonate currently logged-in legitimate Principal and thereby  
2097 gain access to protected resources.

2098           **Description:** Once a Principal is successfully logged into an identity provider, subsequent  
2099 `<AuthnRequest>` messages from different service providers concerning that Principal will  
2100 not necessarily cause the Principal to be reauthenticated. Principals must, however, be  
2101 authenticated unless the identity provider can determine that an `<AuthnRequest>` is  
2102 associated not only with the Principal's identity, but also with a validly authenticated identity  
2103 provider session for that Principal.

2104           **Countermeasure:** In implementations where this threat is a concern, identity providers  
2105 MUST maintain state information concerning active sessions, and MUST validate the  
2106 correspondence between an `<AuthnRequest>` and an active session before issuing an  
2107 `<AuthnResponse>` without first authenticating the Principal. Cookies posted by identity  
2108 providers MAY be used to support this validation process, though Liberty does not mandate a  
2109 cookie-based approach.

#### 2110 **4.4.2.2 Liberty-Enabled Client and Proxy Profile**

2111 **Threat:** Intercepted `<lib:AuthnRequestEnvelope>` and `<lib:AuthnResponse>` and  
2112 subsequent Principal impersonation.

2113 **Description:** A spurious system entity can interject itself as a man-in-the-middle (MITM) between  
2114 the user agent (LECP) and a legitimate serviceprovider, where it acts in the service provider role in  
2115 interactions with the (LECP), and in the user agent role in interactions with the legitimate service  
2116 provider. In this way, as a first step, the MITM is able to intercept the service provider's  
2117 `<lib:AuthnRequestEnvelope>` (step 3 of section 3.2.5) and substitute any URL of its choosing  
2118 for the `<lib:AssertionConsumerServiceURL>` value before forwarding the  
2119 `<lib:AuthnRequestEnvelope>` on to the LECP. Typically, the MITM will insert a URL value  
2120 that points back to itself. Then, if the LECP subsequently receives a  
2121 `<lib:AuthnResponseEnvelope>` from the identity provider (step 6 in section 3.2.5) and  
2122 subsequently sends the contained `<lib:AuthnResponse>` to the  
2123 `<lib:AssertionConsumerServiceURL>` received from the MITM, the MITM will be able to  
2124 masquerade as the Principal at the legitimate service provider.

2125 **Countermeasure:** The identity provider specifies to the LECP the address to which the LECP must  
2126 send the `<lib:AuthnResponse>`. The `<lib:AssertionConsumerServiceURL>` in the  
2127 `<lib:AuthnResponseEnvelope>` element is for this purpose. This URL value is among the

2128 metadata that identity and service providers must exchange in the process of establishing their  
2129 operational relationship (see sections 3.1 and 3.1.3).

#### 2130 4.4.2.3 Federation

2131 **Threat:** Collusion among service providers can violate privacy of the Principal

2132 **Description:** When a group of service providers collude to share the  
2133 `<lib:IDPProvidedNameIdentifier>` of a Principal, they can track and in general  
2134 compromise the privacy of the principal. More generally, this threat exists for any common  
2135 data (e.g. phone number) shared by rogue system entities.

2136 **Countermeasure:** The `<lib:IDPProvidedNameIdentifier>` is required to be unique  
2137 for each identity provider to service provider relationship. However, this requirement does  
2138 not eliminate the threat when there are rogue participants under the Principal's identity  
2139 federation. The only protection is for Principals to be cautious when they choose service  
2140 providers and understand their privacy policies.

2141 **Threat:** Poorly generated name identifiers may compromise privacy

2142 **Description:** The federation protocol mandates that the `<lib:NameIdentifier>` elements  
2143 be unique within a Principal's federated identities. The name identifiers exchanged are  
2144 pseudonyms and, to maintain the privacy of the Principal, should be resistant to guessing or  
2145 derivation attacks.

2146 **Countermeasure:** Name identifiers should be constructed using pseudo-random values that  
2147 have no discernable correspondence with the Principal's identifier (or name) used by the  
2148 entity that generates the name identifier.

#### 2149 4.4.3 Name Registration

2150 No known threats.

#### 2151 4.4.4 Federation Termination: HTTP-Redirect-Based Profile

2152 **Threat:** Attacker can monitor and disrupt termination

2153 **Description:** During the initial steps, a passive attacker can collect the  
2154 `<lib:FederationTerminationNotification>` information when it is issued in the  
2155 redirect. This threat is possible because the first and second steps are not required to use  
2156 `https` as the URL scheme. An active attacker may be able to intercept and modify the  
2157 message conveyed in step 2 because the digital signature only covers a portion of the  
2158 message. This initial exchange also exposes the name identifier. Exposing these data poses a  
2159 privacy threat.

2160 **Countermeasure:** All exchanges should be conducted over a secure transport such as SSL or  
2161 TLS.

#### 2162 4.4.5 Single Logout: HTTP-Redirect-Based Profile

2163 **Threat:** Passive attacker can collect a Principal's name identifier

2164 **Description:** During the initial steps, a passive attacker can collect the  
2165 `<lib:LogoutNotification>` information when it is issued in the redirect. Exposing these  
2166 data poses a privacy threat.

2167           **Countermeasure:** All exchanges should be conducted over a secure transport such as SSL or  
2168           TLS.

2169   **Threat:** Unsigned <lib:LogoutNotification> message

2170           **Description:** An Unsigned <lib:LogoutNotification> could be injected by a spurious  
2171           system entity thus denying service to the Principal. Assuming that the NameIdentifier can  
2172           be deduced or derived then it is conceivable that the user agent could be directed to deliver a  
2173           fabricated <lib:LogoutNotification> message.

2174           **Countermeasure:** Sign the <lib:LogoutNotification> message. The identity provider  
2175           can also verify the identity of a Principal in the absence of a signed request.

#### 2176   **4.4.6     Identity Provider Introduction**

2177   No known threats.

### 2178   **5   References**

- 2179   [Anders]           Rundgren, A., "A suggestion on how to implement SAML browser bindings  
2180           without using Artifacts," [http://www.x-obi.com/OBI400/andersr-browser-](http://www.x-obi.com/OBI400/andersr-browser-artifact.ppt)  
2181           [artifact.ppt](http://www.x-obi.com/OBI400/andersr-browser-artifact.ppt), August 2001.
- 2182   [HTML4]           Raggett, D., Le Hors, A., Jacobs, I., "HTML 4.01 Specification,"  
2183           <http://www.w3.org/TR/html401>, W3C, December 1999.
- 2184   [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)  
2185           [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.
- 2186   [LibertyProtSchema] Beatty, J., "Liberty Protocols and Schemas Specification,"  
2187           [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)  
2188           [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.
- 2189   [Rescorla-Sec]     Rescorla, E., et al., "Guidelines for Writing RFC Text on Security  
2190           Considerations," [http://www.ietf.org/internet-drafts/draft-rescorla-sec-](http://www.ietf.org/internet-drafts/draft-rescorla-sec-cons-04.txt)  
2191           [cons-04.txt](http://www.ietf.org/internet-drafts/draft-rescorla-sec-cons-04.txt), February 2002.
- 2192   [RFC1750]          Eastlake, D., Croker, S., Schiller, J., "Randomness Recommendations for  
2193           Security," <ftp://ftp.isi.edu/in-notes/rfc1750.txt>, RFC 1750, December 1994
- 2194   [RFC1945]          Berners-Lee, T., Fielding, R., Frystyk, H., "Hypertext Transfer Protocol --  
2195           HTTP/1.0," <ftp://ftp.isi.edu/in-notes/rfc1945.txt>, RFC 1945, May 1996.
- 2196   [RFC2045]          Freed, N., Borenstein, N., "Multipurpose Internet Mail Extensions (MIME)  
2197           Part One: Format of Internet Message Bodies," [ftp://ftp.isi.edu/in-](ftp://ftp.isi.edu/in-notes/rfc2045.txt)  
2198           [notes/rfc2045.txt](ftp://ftp.isi.edu/in-notes/rfc2045.txt), RFC 2045, November 1996.
- 2199   [RFC2109]          Kristol, D., Montulli, L., "HTTP State Management Mechanism,"  
2200           <ftp://ftp.isi.edu/in-notes/rfc2109.txt>, RFC 2109, February 1997.
- 2201   [RFC2119]          Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels,"  
2202           <ftp://ftp.isi.edu/in-notes/rfc2119.txt>, RFC 2119, March 1997.
- 2203   [RFC2246]          Dierks, T., Allen, C., "The TLS Protocol Version 1.0," [ftp://ftp.isi.edu/in-](ftp://ftp.isi.edu/in-notes/rfc2246.txt)  
2204           [notes/rfc2246.txt](ftp://ftp.isi.edu/in-notes/rfc2246.txt), RFC 2246, January 1999.
- 2205   [RFC2396]          Berners-Lee, T., Fielding, R., Masinter, L., "Uniform Resource Identifiers  
2206           (URI): Generic Syntax," <ftp://ftp.isi.edu/in-notes/rfc2396.txt>, RFC 2396,  
2207           August 1998.



- 2208 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P.,  
2209 Berners-Lee, T., "Hypertext Transfer Protocol -- HTTP/1.1,"  
2210 <ftp://ftp.isi.edu/in-notes/rfc2616.txt>, RFC 2616, June 1999.
- 2211 [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen,  
2212 A., Stewart, L., "HTTP Authentication: Basic and Digest Access  
2213 Authentication," <ftp://ftp.isi.edu/in-notes/rfc2617.txt>, RFC 2617, June 1999.
- 2214 [RFC2774] Nielsen, H., Leach P., Lawrence, S., "An HTTP Extension Framework,"  
2215 <ftp://ftp.isi.edu/in-notes/rfc2774.txt>, RFC 2774, February 2000.
- 2216 [RFC3106] Eastlake, D., Goldstein, T., "ECML v1.1: Field Specifications for E-  
2217 Commerce," <ftp://ftp.isi.edu/in-notes/rfc3106.txt>, RFC 3106, April 2001
- 2218 [SAMLBind] Mishra, P., Ed., "Bindings and Profiles for the OASIS Security Assertion  
2219 Markup Language (SAML)," [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/cs-sstc-bindings-01.pdf)  
2220 [open.org/committees/security/docs/cs-sstc-bindings-01.pdf](http://www.oasis-open.org/committees/security/docs/cs-sstc-bindings-01.pdf), OASIS, May  
2221 2002.
- 2222 [SAMLCore] Hallam-Baker, P., Maler, e., Eds.), "Assertions and Protocol for the OASIS  
2223 Security Assertion Markup Language (SAML)," [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/cs-sstc-core-01.pdf)  
2224 [open.org/committees/security/docs/cs-sstc-core-01.pdf](http://www.oasis-open.org/committees/security/docs/cs-sstc-core-01.pdf). OASIS, May 2002.
- 2225 [SAMLGloss] Hodges, J., Maler, E, Eds., "Glossary for the OASIS Security Assertion  
2226 Markup Language (SAML)," [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/cs-sstc-glossary-01.pdf)  
2227 [open.org/committees/security/docs/cs-sstc-glossary-01.pdf](http://www.oasis-open.org/committees/security/docs/cs-sstc-glossary-01.pdf), OASIS, May  
2228 2002.
- 2229 [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)  
2230 [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,  
2231 December 2001.
- 2232 [SAMLSec] McLaren, C., Ed., "Security Considerations for the OASIS Security Assertion  
2233 Markup Language (SAML)," [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/cs-sstc-sec-consider-01.pdf)  
2234 [open.org/committees/security/docs/cs-sstc-sec-consider-01.pdf](http://www.oasis-open.org/committees/security/docs/cs-sstc-sec-consider-01.pdf), OASIS, May  
2235 2001.
- 2236 [Schema1] Thompson, H. S., et al., "XML Schema Part 1: Structures,"  
2237 <http://www.w3.org/TR/xmlschema-1/>, W3C Recommendation, May 2001.
- 2238 [Schema2] Biron, P. V., et al., "XML Schema Part 2: Datatypes,"  
2239 <http://www.w3.org/TR/xmlschema-2/>, W3C Recommendation, May 2001.
- 2240 [ShibMarlena] Erdos, M., "Shibboleth Architecture DRAFT v1.1,"  
2241 [http://middleware.internet2.edu/shibboleth/docs/draft-erdos-shibboleth-](http://middleware.internet2.edu/shibboleth/docs/draft-erdos-shibboleth-architecture1-01.pdf)  
2242 [architecture1-01.pdf](http://middleware.internet2.edu/shibboleth/docs/draft-erdos-shibboleth-architecture1-01.pdf), June 2001.
- 2243 [SOAP1.1] Box, D., et al., "Simple Object Access Protocol (SOAP) 1.1,"  
2244 <http://www.w3.org/TR/SOAP>, W3C Note, May 2000.
- 2245 [SSLv3] Freier, A. O., Karlton, P., Kocher, P., "The SSL Protocol Version 3.0,"  
2246 <http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt>, Internet  
2247 Draft, November 1996.
- 2248 [WML1.3] "Wireless Application Protocol Wireless Markup Language Specification  
2249 Version 1.3," Wireless Application Protocol Forum, Ltd.,  
2250 <http://www.wapforum.org/>, February 2000
- 2251 [XMLSig] Eastlake, D., Reagle, J., Solo, D., "XML-Signature Syntax and Processing,"  
2252 <http://www.w3.org/TR/xmlsig-core/>, W3C Recommendation, February  
2253 2002.

2254