



Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification

Version: v2.0

Editors:

Jeff Hodges, NeuStar, Inc.
Robert Aarts, Hewlett-Packard
Paul Madsen, NTT
Scott Cantor, Internet2 / The Ohio State University

Contributors:

Conor Cahill, America Online, Inc.
Darryl Champagne, IEEE-ISTO
Gary Ellison, Sun Microsystems, Inc.
Greg Whitehead, Hewlett-Packard

Abstract:

Abstract

This specification defines an ID-WSF Authentication Protocol based on a profile of the Simple Authentication and Security Layer (SASL) framework mapped onto ID-* SOAP-bound messages. It also defines an ID-WSF Authentication Service which Identity Providers may offer. This service is based on the authentication protocol. The authentication service enables Web Services Consumers and/or Liberty-enabled User Agents or Devices to authenticate with Identity Providers, using various authentication mechanisms, and obtain ID-WSF security tokens. Next, it defines the ID-WSF Single Sign-On Service, which provides SAML authentication assertions to Web Service Consumers via profiles of the SAML 2.0 Authentication Request protocol, enabling Web Service Consumers and/or Liberty-enabled User Agents or Devices to interact with SAML-based services. Finally, it defines the ID-WSF Identity Mapping Service, which allows Web Service Consumers to obtain identity tokens for use in web service invocations and referencing principals while preserving privacy.

Filename: liberty-idwsf-authn-svc-v2.0.pdf

1 **Notice**

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

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

15 Copyright © 2006 Adobe Systems; America Online, Inc.; American Express Company; Amsoft Systems Pvt Ltd.;
16 Avatier Corporation; Axalto; Bank of America Corporation; BIPAC; BMC Software, Inc.; Computer Associates
17 International, Inc.; DataPower Technology, Inc.; Diversinet Corp.; Enosis Group LLC; Entrust, Inc.; Epok, Inc.;
18 Ericsson; Fidelity Investments; Forum Systems, Inc.; France Télécom; French Government Agence pour le
19 développement de l'administration électronique (ADAE); Gamefederation; Gemplus; General Motors; Giesecke &
20 Devrient GmbH; GSA Office of Governmentwide Policy; Hewlett-Packard Company; IBM Corporation; Intel
21 Corporation; Intuit Inc.; Kantega; Kayak Interactive; MasterCard International; Mobile Telephone Networks (Pty)
22 Ltd; NEC Corporation; Netegrity, Inc.; NeuStar, Inc.; Nippon Telegraph and Telephone Corporation; Nokia
23 Corporation; Novell, Inc.; NTT DoCoMo, Inc.; OpenNetwork; Oracle Corporation; Ping Identity Corporation;
24 Reactivity Inc.; Royal Mail Group plc; RSA Security Inc.; SAP AG; Senforce; Sharp Laboratories of America;
25 Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Supremacy Financial Corporation; Symlabs, Inc.;
26 Telecom Italia S.p.A.; Telefónica Móviles, S.A.; Trusted Network Technologies; UTI; VeriSign, Inc.; Vodafone
27 Group Plc.; Wave Systems Corp. All rights reserved.

28 Liberty Alliance Project
29 Licensing Administrator
30 c/o IEEE-ISTO
31 445 Hoes Lane
32 Piscataway, NJ 08855-1331, USA
33 info@projectliberty.org

34 Contents

35	1. Introduction	5
36	2. Notation and Conventions	6
37	2.1. Requirements Keywords	6
38	2.2. XML Namespaces	6
39	3. Terminology	7
40	4. Authentication Protocol	10
41	4.1. Conceptual Model	10
42	4.2. Schema Declarations	10
43	4.3. SOAP Header Blocks and SOAP Binding	10
44	4.3.1. SOAP Binding	10
45	4.4. SASL Profile Particulars	10
46	4.4.1. SASL "Service Name"	11
47	4.4.2. Composition of SASL Mechanism Names	11
48	4.5. Authentication Exchange Security	11
49	4.6. Protocol Messages	11
50	4.6.1. The <SASLRequest> Message	11
51	4.6.2. The <SASLResponse> Message	14
52	4.7. Sequencing of the Authentication Exchange	17
53	5. Authentication Service	21
54	5.1. Conceptual Model	21
55	5.1.1. Stipulating a Particular Authentication Context	21
56	5.2. URI Declarations	21
57	5.3. Rules for Authentication Service Providers	22
58	5.4. Rules for Authentication Service Consumers	23
59	5.5. Authentication Service Interaction Example	23
60	6. Single Sign-On Service	26
61	6.1. Conceptual Model	26
62	6.2. Single Sign-On Service URIs	26
63	6.3. ID-WSF Enhanced Client or Proxy SSO Profile	27
64	6.3.1. Profile Overview	27
65	6.3.2. Profile Description	27
66	6.4. ID-WSF SAML Token Service Profile	28
67	6.4.1. Profile Overview	28
68	6.4.2. Profile Description	29
69	6.4.3. Use of SAML 2.0 Authentication Request Protocol	29
70	6.5. Use of Metadata	31
71	6.6. Inclusion of ID-WSF Endpoint References	31
72	7. Identity Mapping Service	32
73	7.1. Conceptual Model	32
74	7.2. Schema Declarations	32
75	7.3. SOAP Binding	32
76	7.3.1. Identity Mapping Service URIs	32
77	7.4. Protocol Messages and Usage	33
78	7.4.1. Element <IdentityMappingRequest>	33
79	7.4.2. Element <IdentityMappingResponse>	34
80	7.5. SAML Identity Tokens	36
81	7.5.1. Assertions	36
82	7.5.2. Identifiers	36
83	7.6. Security and Privacy Considerations	37
84	7.7. Example Identity Mapping Exchange	37
85	8. Password Transformations: The PasswordTransforms Element	39
86	9. Acknowledgments	41

87	References	42
88	A. Listing of Simple Authentication and Security Layer (SASL) Mechanisms	45
89	B. Password Transformations	47
90	1. Truncation	47
91	2. Lowercase	47
92	3. Uppercase	47
93	4. Select	47
94	C. liberty-idwsf-authn-svc-v2.0.xsd Schema Listing	49
95	D. liberty-idwsf-idmapping-svc-v2.0.xsd Schema Listing	52
96	E. liberty-idwsf-utility-v2.0.xsd Schema Listing	54
97	F. liberty-idwsf-authn-svc-v2.0.wsdl WSDL Listing	56
98	G. liberty-idwsf-sso-svc-v2.0.wsdl WSDL Listing	58
99	H. liberty-idwsf-idmapping-svc-v2.0.wsdl WSDL Listing	60

100 **1. Introduction**

101 The Simple Object Access Protocol (SOAP) specifications, [[SOAPv1.1](#)] and [[SOAPv1.2](#)], define an XML-based
102 [[XML](#)] messaging paradigm, but do not specify any particular security mechanisms. They do not, in particular,
103 describe how one *SOAP node* may authenticate with another *SOAP node* via an exchange of SOAP messages. Thus
104 it is left to SOAP-based web services frameworks to provide their own notions of security, such as defining how
105 authentication is accomplished.

106 This specification defines how to perform *general identity authentication* [[WooLam92](#)], also known as *peer entity*
107 *authentication* [[RFC2828](#)], over SOAP, in the context of the Liberty Identity Web Services Framework (ID-WSF)
108 [[LibertyIDWSFOverview](#)]. Rather than specify the particulars of one or more *authentication mechanisms* directly in
109 this specification, we profile the Simple Authentication and Security Layer (SASL) framework [[RFC4422](#)].

110 SASL is an approach to modularizing protocol *design* such that the security design components, e.g. authentication
111 and security layer mechanisms, are reduced to a uniform abstract interface. This facilitates a protocol's use of an open-
112 ended set of security mechanisms, as well as a so-called "late binding" between implementations of the protocol and
113 the security mechanisms' implementations. This late binding can occur at implementation- and/or deployment-time.
114 The SASL specification also defines how one packages authentication and security layer mechanisms to fit into the
115 SASL framework, where they are known as *SASL mechanisms*, as well as register them with the Internet Assigned
116 Numbers Authority (IANA) [[IANA](#)] for reuse.

117 This specification is organized as follows. First, it defines the ID-WSF Authentication Protocol. Then, it defines
118 an ID-WSF Authentication Service Identity Providers may offer, which is based on the authentication protocol. This
119 authentication service enables Web Services Consumers and/or Liberty-enabled User Agents or Devices to authenticate
120 with Identity Providers using various authentication mechanisms and obtain ID-WSF security tokens. Next, it defines
121 the ID-WSF Single Sign-On Service, which provides SAML authentication assertions to Web Service Consumers via
122 profiles of the SAML 2.0 Authentication Request protocol, enabling Web Service Consumers and/or Liberty-enabled
123 User Agents or Devices to interact with SAML-based services. Finally, it defines the ID-WSF Identity Mapping
124 Service, which allows Web Service Consumers to obtain identity tokens for use in web service invocations and
125 referencing principals while preserving privacy.

126 **2. Notation and Conventions**

127 This specification uses schema documents conforming to W3C XML Schema [Schema1-2] and normative text to
 128 describe the syntax and semantics of XML-encoded protocol messages.

129 **2.1. Requirements Keywords**

130 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT",
 131 "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]:

132 “they MUST only be used where it is actually required for interoperation or to limit behavior which
 133 has potential for causing harm (e.g., limiting retransmissions)”

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

137 **2.2. XML Namespaces**

138 This specification uses the XML namespace prefixes listed in Table 1.

139 **Table 1. XML Namespaces used in this specification**

Prefix	Namespace
sa:	Represents the ID-WSF Authentication Service namespace: urn:liberty:sa:2006-08 Note This is the point of definition of this namespace. This namespace is the default for instance fragments, type names, and element names in this document when a namespace is not explicitly noted.
disco:	Represents the namespace defined in [LibertyDisco].
sec:	Represents the namespace defined in [LibertySecMech].
md:	Represents the namespace defined in [SAMLMeta2].
pp:	Represents the namespace defined in [LibertyIDPP].
s:	Represents the SOAP namespace: http://www.w3.org/2001/12/soap-envelope , defined in [SOAPv1.1].
saml2:	Represents the SAML V2.0 Assertion namespace defined in [SAMLCore2]
samlp2:	Represents the SAML V2.0 Protocol namespace defined in [SAMLCore2]
sb:	Represents the Liberty namespace defined in [LibertySOAPBinding]
lu:	Represents the Liberty ID-WSF utility namespace (see Appendix E).
xs:	Represents the W3C XML schema namespace (http://www.w3.org/2001/XMLSchema) defined in [Schema1-2].

140 3. Terminology

141 This section defines key terminology used in this specification. Definitions for these, as well as other Liberty-specific
142 terms, may also be found in [[LibertyGlossary](#)]. Note that the definition of some terms below differ slightly from the
143 definition given in [[LibertyGlossary](#)]. For example see the definitions for *client* and *server*. This is because in such
144 cases, the definition given in [[LibertyGlossary](#)] is a more general one, and the definition given here is a narrower one,
145 specific to the context of this specification. See also [[RFC2828](#)] for overall definitions of security-related terms, in
146 general. Other specific references are also cited below.

147 authentication *Authentication* is the process of confirming a *system entity*'s asserted *identity* with a specified,
148 or understood, level of confidence [[TrustInCyberspace](#)].

149 authentication assertion A *SAML assertion* typically consisting of a single <AuthenticationStatement>. The
150 assertion issuer is stating that the subject of the assertion authenticated with it at some point
151 in time. Assertions are typically time-limited [[SAMLCore2](#)].

152 authentication exchange See *authentication protocol exchange*.

153 authentication mechanism An *authentication mechanism* is a particular, identifiable, process or technique that
154 results in a confirmation of a *system entity*'s asserted identity with a specified, or understood,
155 level of confidence.

156 authentication protocol exchange *Authentication protocol exchange* is the term used in [[RFC4422](#)] to refer to the
157 sequence of messages exchanged between the *client* and *server* as specified and governed by
158 a particular *SASL mechanism* being employed to effect an act of *authentication*.

159 authentication server The precise, specific *role* played by a *server* in the protocol message exchanges defined in
160 this specification.

161 Authentication Service (AS) Short form of "ID-WSF Authentication Service". The AS is a discoverable ID-WSF
162 service.

163 Authentication Service Consumer A *Web Service Consumer* (WSC) implementing the *client*-side of the ID-WSF
164 Authentication Protocol (which is defined in this specification).

165 Authentication Service Provider (AS Provider) A *Web Service Provider* (WSP) implementing the *server*-side of
166 the ID-WSF Authentication Service defined in this specification ([Section 5: Authentication](#)
167 [Service](#)).

168 client A *role* assumed by a *system entity* who either explicitly or implicitly initiates an authentica-
169 tion exchange [[RFC2828](#)]. *Client* is implicitly defined in [[RFC4422](#)]. Also known as a *SASL*
170 *client*.

171 discoverable A *discoverable* "in principle" service is one having a *service type URI* assigned (this is
172 typically in done in the specification defining the service). A discoverable "in practice"
173 service is one that is registered in some discovery service instance.

174 ID-WSF services are by definition discoverable "in principle" because such services are
175 assigned a *service type URI* facilitating their registration in *Discovery Service* instances.

176 final SASL response The final <SASLResponse> message sent from the *server* to the *client* in an *authentication*
177 *exchange*.

178	ID-WSF EPR	An <i>ID-WSF Endpoint Reference</i> is a reference to a <i>service instance</i> . It contains the address, security context, and other metadata necessary for contacting the identified service instance.
179		The underlying structure of an ID-WSF EPR is based on the <i>wsa:EndpointReference</i> of [WSAv1.0-SOAP] [WSAv1.0].
180		
181		
182	initial response	A [RFC4422] term referring to <i>authentication exchange data</i> sent by the <i>client</i> in the <i>initial SASL request</i> . It is used by a subset of SASL mechanisms. See Section 5.1 of [RFC4422].
183		
184	initial SASL request	The initial <SASLRequest> message sent from the <i>client</i> to the <i>server</i> in an <i>authentication exchange</i> .
185		
186	(LUAD-)WSC	A <i>Web Service Consumer</i> (WSC) that may or may not also be a <i>Liberty-enabled User Agent or Device</i> .
187		
188	mechanism	A process or technique for achieving a result [Merriam-Webster].
189	message thread	A <i>message thread</i> is a synchronous exchange of messages in a request-response <i>MEP</i> between two <i>SOAP nodes</i> . All the messages of a given message thread are "linked" via each message's <wsa:RelatesTo> header block value being set, by the sender, from the previous successfully received message's <wsa:MessageID> header block value.
190		
191		
192		
193	requester	A <i>system entity</i> which sends a <i>service request</i> to a <i>provider</i> .
194	role	A function or part performed, especially in a particular operation or process [Merriam-Webster].
195		
196	SASL mechanism	A <i>SASL mechanism</i> is an <i>authentication mechanism</i> that has been profiled for use in the context of the <i>SASL framework</i> [RFC4422]. See [RFC2444] for a particular example of profiling an existing authentication mechanism—one-time passwords [RFC2289]—for use in the SASL context. SASL mechanisms are "named"; Mechanism names are listed in the column labeled as "MECHANISMS" in [SASLReg] (a copy of this registry document is reproduced in Appendix A for informational convenience; implementors should always fetch the most recent revision directly from [IANA]).
197		
198		
199		
200		
201		
202		
203	server	A <i>role</i> donned by a <i>system entity</i> which is intended to engage in defined exchanges with <i>clients</i> . This term is implicitly defined in [RFC4422] and in this specification is always synonymous with <i>authentication server</i> .
204		
205		
206	service instance	The physical instantiation of a service. A service instance is a web service at a distinct endpoint.
207		
208	Service Provider (SP)	(1)A <i>role</i> donned by <i>system entities</i> . In the Liberty architecture, <i>Service Providers</i> interact with other system entities primarily via vanilla HTTP.
209		
210		(2) From a Principal's perspective, a Service Provider is typically a website providing services and/or goods.
211		
212	SOAP header block	A [SOAPv1.2] term meaning: An [element] used to delimit data that logically constitutes a single computational unit within the SOAP header. In [SOAPv1.1] these are known as simply <i>SOAP headers</i> , or simply <i>headers</i> . This specification borrows the SOAPv1.2 terminology.
213		
214		
215		
216	SOAP node	A [SOAPv1.2] term describing <i>system entities</i> who are parties to SOAP-based message exchanges that are, for purposes of this specification, also the ultimate destination of the exchanged messages, i.e. <i>SOAP endpoints</i> . In [SOAPv1.1], SOAP nodes are referred to as <i>SOAP endpoints</i> , or simply <i>endpoints</i> . This specification borrows the SOAPv1.2 terminology.
217		
218		
219		

220	system entity	An active element of a computer/network system. For example, an automated process or set of processes, a subsystem, a person or group of persons that incorporates a distinct set of functionality [SAMLGloss2].
221		
222		
223	user identifier	AKA <i>user name</i> or <i>Principal</i> .
224	web service	Generically, a <i>service</i> defined in terms of an <i>XML</i> -based protocol, often transported over <i>SOAP</i> , and/or a service whose instances, and possibly data objects managed therein, are concisely addressable via <i>URIs</i> . As specifically used in Liberty specifications, usually in terms of <i>WSCs</i> and <i>WSPs</i> , it means a web service that's defined in terms of the <i>ID</i> -* "stack", and thus utilizes [LibertySOAPBinding], [LibertySecMech], and is "discoverable" [LibertyDisco].
225		
226		
227		
228		
229		
230	Web Service Consumer	A <i>role</i> donned by a <i>system entity</i> when it makes a request to a <i>web service</i> .
231	Web Service Provider	A <i>role</i> donned by a <i>system entity</i> when it provides a <i>web service</i> .

232 4. Authentication Protocol

233 This section defines the ID-WSF Authentication Protocol. This protocol facilitates authentication between two ID-*
234 entities, and is a profile of SASL [RFC4422].

235 4.1. Conceptual Model

236 The conceptual model for the ID-WSF Authentication Protocol is as follows: an ID-WSF *system entity*, acting in a
237 *Web Services Consumer (WSC) role*, makes an authentication request to another ID-WSF system entity, acting in a
238 *Web Service Provider (WSP) role*, and if the WSP is willing and able, an authentication exchange will ensue.

239 The authentication exchange is comprised of SOAP-bound ID-* messages [LibertySOAPBinding], and can involve an
240 arbitrary number of round trips, dictated by the particular SASL mechanism employed [RFC4422]. The WSC may
241 have out-of-band knowledge of the server's supported SASL mechanisms, or it may send the server its own list of
242 supported SASL mechanisms and allow the server to choose one from among them.

243 At the end of this exchange of messages, the WSC will either be authenticated or not, the nature of the authentication
244 depending upon the SASL mechanism that was employed. Also depending on the SASL mechanism employed, the
245 WSP may be authenticated as well.

246 Other particulars, such as how the WSC knows which WSP to contact for authentication, are addressed below in
247 [Section 6: Single Sign-On Service](#).

248 Note

249 This document does not specify the use of SASL security layers.

250 4.2. Schema Declarations

251 The XML schema [Schema1-2] normatively defined in this section is constituted in the XML Schema file:
252 `liberty-idwsf-authn-svc-v2.0.xsd`, entitled " [Liberty ID-WSF Authentication Service XSD v2.0](#) " (see
253 [Appendix C](#)).

254 Additionally, [Liberty ID-WSF Authentication Service XSD v2.0](#) imports items from `liberty-idwsf-utility-v2.0.xsd`
255 (see [Appendix E: Liberty ID-WSF Utility XSD v2.0](#)), and also from `saml-schema-protocol-2.0.xsd` (see
256 [\[SAMLCore2\]](#)).

257 4.3. SOAP Header Blocks and SOAP Binding

258 This specification does not define any SOAP header blocks. [Section 4.3.1](#), below, constitutes the SOAP binding
259 statement for this specification.

260 4.3.1. SOAP Binding

261 The messages defined below in [Section 4.6](#), e.g. `<SASLRequest>`, are *ordinary ID-* messages* as defined in
262 [\[LibertySOAPBinding\]](#). They are intended to be bound to the [\[SOAPv1.1\]](#) protocol by mapping them directly
263 into the `<s:Body>` element of the `<s:Envelope>` element comprising a SOAP message. [\[LibertySOAPBinding\]](#)
264 normatively specifies this binding.

265 **Note**

266 Implementations of this specification MUST use the "Messaging-specific Header Blocks", as specified in [LibertySOAPBinding], to establish a *message thread* and thus correlate their authentication exchanges. See Section 5.5: Authentication Service Interaction Example for an example.

269 **4.4. SASL Profile Particulars**

270 The ID-WSF Authentication Protocol is based on SASL [RFC4422], and thus "profiles" SASL. Section 4 of [RFC4422] specifies SASL's "profiling requirements". This section of this specification addresses some particulars of profiling SASL that are not otherwise addressed in the sections defining the protocol messages (Section 4.6: Protocol Messages), and their sequencing (Section 4.7: Sequencing of the Authentication Exchange).

274 **4.4.1. SASL "Service Name"**

275 The SASL "Service Name" specified herein is: **idwsf**

276 **4.4.2. Composition of SASL Mechanism Names**

277 The protocol messages defined below at times convey a SASL mechanism name, or a list of SASL mechanism names, as values of message element attributes.

279 These mechanism names are typically taken from the column labeled as "MECHANISMS" in [SASLReg], but MAY be site-specific.

281 These names, and lists of these names, MUST follow these rules:

- 282 • The character composition of a SASL mechanism name MUST be as defined in [IANA]'s SASL Mechanism Registry [SASLReg].
- 284 • A list of SASL mechanism names MUST be composed of names as defined above, separated by ASCII space chars (hex "20").

286 **4.5. Authentication Exchange Security**

287 This authentication protocol features the flexibility of having implementations being able to select at runtime the actual authentication mechanism (aka SASL mechanism) to employ. This however may introduce various vulnerabilities depending on the actual mechanism employed. Some mechanisms may be vulnerable to passive and/or active attacks. Also, since the server selects the SASL mechanism from a list supplied by the client, a compromised server, or a man-in-the-middle, can cause the weakest mechanism offered by the client to be employed.

292 Thus it is RECOMMENDED that the authentication protocol exchange defined herein (Section 4.7: Sequencing of the Authentication Exchange) be employed over a TLS/SSL channel [RFC4346] as amended by [RFC4366]. This will ensure the integrity and confidentiality of the authentication protocol messages. Additionally, clients SHOULD authenticate the server via TLS/SSL validation procedures. This will help guard against man-in-the-middle attacks.

296 **4.6. Protocol Messages**

297 This section defines the protocol's messages, along with their message element attribute values, and their semantics. The sequencing of protocol interactions, also known as the *authentication exchange*, is defined below in Section 4.7: Sequencing of the Authentication Exchange .

300 **4.6.1. The <SASLRequest> Message**

301 Figure 1 shows the schema fragment from Liberty ID-WSF Authentication Service XSD v2.0 describing the <SASLRequest> message. This message has the following attributes:

- 303 • **mechanism** [Required] — Used to convey a list of one-or-more client-supported SASL mechanism names to the
 304 server, or to signal the server if the client wishes to abort the exchange. It is included on all <SASLRequest>
 305 messages sent by the client.
- 306 • **authzID** [Optional] — The authzID, also known as *user identifier* or *username* or *Principal*, that the client
 307 wishes to establish as the "authorization identity" per [RFC4422].
- 308 • **advisoryAuthnID** [Optional] — The advisoryAuthnID may be used to advise the server what authentication
 309 identity will be asserted by the client via the selected SASL mechanism; i.e. it is a "hint". The advisoryAuthnID
 310 provides a means for server implementations to optimize their behavior on a per authentication identity basis.
 311 E.g. if a client requests to execute a certain SASL mechanism on behalf of some given authentication identity
 312 (represented by advisoryAuthnID) and authorization identity (represented by authzID) pair, the server can
 313 decide whether to proceed without having to execute the SASL mechanism (execution of which might involve
 314 more than a single round-trip). Server implementations that make use of the optional advisoryAuthnID attribute
 315 SHOULD be capable of processing initial <SASLRequest> messages that do not include the advisoryAuthnID
 316 attribute.
- 317 • **Any Attributes** [Optional] — Zero or more extension attributes qualified by an XML namespace other than the
 318 Authentication Service namespace.

```

319
320 <xs:element name="SASLRequest">
321   <xs:complexType>
322     <xs:sequence>
323
324       <xs:element name="Data" minOccurs="0">
325         <xs:complexType>
326           <xs:simpleContent>
327             <xs:extension base="xs:base64Binary" />
328           </xs:simpleContent>
329         </xs:complexType>
330       </xs:element>
331
332       <xs:element ref="sampl2:RequestedAuthnContext" minOccurs="0" />
333
334       <xs:element name="Extensions" minOccurs="0">
335         <xs:complexType>
336           <xs:sequence>
337             <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded" />
338           </xs:sequence>
339         </xs:complexType>
340       </xs:element>
341
342     </xs:sequence>
343
344     <xs:attribute name="mechanism"
345       type="xs:string"
346       use="required" />
347
348     <xs:attribute name="authzID"
349       type="xs:string"
350       use="optional" />
351
352     <xs:attribute name="advisoryAuthnID"
353       type="xs:string"
354       use="optional" />
355
356     <xs:anyAttribute namespace="##other" processContents="lax" />
357
358   </xs:complexType>
359 </xs:element>

```

360 **Figure 1. <SASLRequest> Message Element — Schema Fragment**

361 The <SASLRequest> message has the following sub-elements:

- 362 • **<Data>** — This element is used by the client to send SASL mechanism data to the server. In [RFC4422] parlance,
363 this data is termed a "client response". Its content model is base64-encoded data.
- 364 • **<samlp2:RequestedAuthnContext>** — This element is used by the client to convey to the server a desired
365 authentication context. It is used only on the initial SASL request (see Section 4.7: Sequencing of the Au-
366 thentication Exchange). If present, the server uses the information in the <samlp2:RequestedAuthnContext>
367 in combination with mechanism attribute when choosing the SASL mechanism to execute. The background use
368 case for <samlp2:RequestedAuthnContext> is presented in Section 5.1: Authentication Service: Conceptual
369 Model. See also: [LibertyAuthnContext] and [LibertyProtSchema].
- 370 • **<Extensions>** — This contains optional request extensions that are agreed upon between the client and server.
371 Extension elements MUST be namespace-qualified by a non-AS namespace.

```
372
373 <?xml version="1.0" encoding="UTF-8"?>
374
375 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
376           xmlns:sa="urn:liberty:sa:2006-08"
377           xmlns:sb="urn:liberty:wsf:soap-bind:1.0"
378           xmlns:pp="urn:liberty:id-sis-pp:2003-08">
379
380   <S:Header>
381
382     <!-- various header blocks, as defined in the
383          SOAP Binding spec, go here -->
384
385   </S:Header>
386
387   <S:Body>
388
389     <sa:SASLRequest sa:mechanism="foo">
390       <sa:Data>
391         qwyGHhSWpjQu5yq.....vUUlVONmOZtfzgFz
392       <sa:Data>
393     </sa:SASLRequest>
394
395   </S:Body>
396
397 </S:Envelope>
398
```

399 **Example 1. A SASLRequest Bound into a SOAP Message**

400 **4.6.1.1. <SASLRequest> Usage**

401 The <SASLRequest> message is used to initially convey to the server a:

- 402 • list of one or more client-supported SASL mechanism names,
403 ..in combination with optional:
- 404 • authzID attribute, and/or,
405 • advisoryAuthnID attribute, and/or,
406 • <samlp2:RequestedAuthnContext> element.

407 In the case where a single SASL mechanism name is conveyed, the <SASLRequest> message can contain a so-called
408 *initial response* (see Section 5 of [RFC4422]) in the <Data> element.

409 If the server's subsequent <SASLResponse> message signals that the authentication exchange should continue—and
410 thus contains a server "challenge"—the client will send another <SASLRequest> message, with the <Data> element
411 containing the client's "response" to the challenge. This sequence of server challenges and client responses continues
412 until the server signals a successful completion or aborts the exchange.

413 The *mechanism* attribute is used in these intermediate <SASLRequest> messages to signal the client's intentions to
414 the server. This is summarized in the next section.

415 [Section 4.7: Sequencing of the Authentication Exchange](#), in combination with the next section, normatively defines
416 the precise <SASLRequest> message format as a function of the sequencing of the authentication exchange.

417 **4.6.1.2. Values for *mechanism* attribute of <SASLRequest>**

418 The list below defines the allowable values for the *mechanism* attribute of the <SASLRequest> message element,
419 and the resulting message semantics.

420 **Note**

421 In items #2 and #1, the *mechanism* attribute contains one or more SASL mechanism names, respectively. The rules
422 noted in [Section 4.4.2: Composition of SASL Mechanism Names](#) MUST be adhered to in such cases.

423 1. **Multiple SASL mechanism names** — See [Example 2](#). In this case, the <SASLRequest> message MUST NOT
424 contain any "initial response" data, and MUST be the initial SASL request. See [Section 4.6.2.1.2](#) for details on
425 the returned <SASLResponse> message in this case.

```
426  
427     <SASLRequest mechanism="GSSAPI OTP PLAIN" />  
428
```

429 **Example 2. <SASLRequest> Specifying Multiple Client-supported Mechanism Names**

430 2. **A single SASL mechanism name** — In this case, the <SASLRequest> message MAY contain *initial response*
431 data. See [Example 3](#).

```
432  
433     <SASLRequest mechanism="GSSAPI">  
434       <Data>  
435         Q29ub3IgcQ2FoaWxsIGNhc3VhbGx5IGlhbmdsZXMgcGFzc3dvcmRzCg==  
436       </Data>  
437     </SASLRequest>  
438
```

439 **Example 3. <SASLRequest> Specifying a Single Mechanism Name**

440 3. **A NULL string ("")** — This indicates to the authentication server that the client wishes to abort the authentica-
441 tion exchange. See [Example 4](#).

```
442  
443     <SASLRequest mechanism="" />  
444
```

445 **Example 4. <SASLRequest> Message Aborting the SASL Authentication Exchange**

446 4.6.2. The <SASLResponse> Message

447 Figure 2 shows the schema fragment from [Liberty ID-WSF Authentication Service XSD v2.0](#) describing the
448 <SASLResponse> message. This message has the following attributes:

- 449 • **serverMechanism** [Optional] — The server's choice of SASL mechanism from among the list sent by the client.
- 450 • **Any Attributes** [Optional] — Zero or more extension attributes qualified by an XML namespace other than the
451 Authentication Service namespace.

```
452 <xs:element name="SASLResponse">
453   <xs:complexType>
454     <xs:sequence>
455       <xs:element ref="Status"/>
456
457       <xs:element ref="PasswordTransforms" minOccurs="0"/>
458
459       <xs:element name="Data" minOccurs="0">
460         <xs:complexType>
461           <xs:simpleContent>
462             <xs:extension base="xs:base64Binary"/>
463           </xs:simpleContent>
464         </xs:complexType>
465       </xs:element>
466
467       <!-- ID-WSF EPRs -->
468       <xs:element ref="wsa:EndpointReference"
469         minOccurs="0"
470         maxOccurs="unbounded"/>
471
472     </xs:sequence>
473
474     <xs:attribute name="serverMechanism"
475       type="xs:string"
476       use="optional"/>
477
478     <xs:anyAttribute namespace="##other" processContents="lax"/>
479
480   </xs:complexType>
481 </xs:element>
```

485 **Figure 2. <SASLResponse> Message Element - Schema Fragment**

486 The <SASLResponse> message has the following sub-elements:

- 487 • **<Status>** — This element is from [Liberty ID-WSF Utility XSD v2.0](#) and is used to convey status from the
488 server to the client. See below.
- 489 • **<PasswordTransforms>** — This element is used to convey to the client any required password transformations.
490 See [Section 8: Password Transformations: The PasswordTransforms Element](#).
- 491 • **<Data>** — This element is used to return SASL mechanism data to the client. Its content model is base64-encoded
492 data.
- 493 • **<wsa:EndpointReference>** — This element is to convey to the client an ID-WSF EPR for the server, in its
494 role as a WSP, upon a successful authentication exchange completion. Multiple instances of it may be used to
495 also convey ID-WSF EPRs for additional instances of other services. Note that any credentials returned as a result
496 of a successful authentication exchange are conveyed within any returned ID-WSF EPRs [[LibertyDisco](#)]. See
497 [Section 5: Authentication Service](#).

498 **4.6.2.1. <SASLResponse> Usage**

499 This message is sent by the server in response to a client <SASLRequest> message. It is used to convey "server
 500 challenges", in [RFC4422] parlance, to the client during an authentication exchange. So-called "client responses"
 501 are correspondingly conveyed to the server via the <SASLRequest> message, defined above. A given authentication
 502 exchange may occur in one "round-trip", or it may involve several round-trips. This depends on the SASL mechanism
 503 being executed.

504 The first <SASLResponse> sent by the server within an authentication exchange (as determined by the particular
 505 authentication mechanism being used) is explicitly distinguished from subsequent <SASLResponse> messages in
 506 terms of child elements and attributes. The final <SASLResponse> sent by the server in an authentication exchange
 507 is similarly distinguished, although with its own particular characteristics. These details are specified below in
 508 [Section 4.7: Sequencing of the Authentication Exchange](#) .

509 It is possible for different authentication mechanisms to be sequenced, the client authenticating to the server with
 510 one after another. For example, after a principal is authenticated with name and password (e.g. with PLAIN or
 511 CRAM-MD5), the service may (because of service or user policy) require additional authentication with SECUR-ID.
 512 Consequently, client implementations should be prepared for a message from the service with a "Continue" status code
 513 but a different "serviceMechanism" than that established in the previous authentication exchange. The message from
 514 the service that indicates such subsequent SASL mechanism may contain a <Data> element intended for processing by
 515 an implementation of the new mechanism. The client should process this message as specified in step 5 of [Section 4.7:](#)
 516 [Sequencing of the Authentication Exchange](#) .

517 The <Status> element (see [Figure 3](#)) is used to convey the authentication server's assessment of the status of the
 518 authentication exchange to the client, via the code attribute (the <Status> element is declared in the [Liberty ID-](#)
 519 [WSF Utility XSD v2.0](#)).

```

520 <xs:complexType name="StatusType">
521   <xs:annotation>
522     <xs:documentation>
523       A type that may be used for status codes.
524     </xs:documentation>
525   </xs:annotation>
526   <xs:sequence>
527     <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
528   </xs:sequence>
529   <xs:attribute name="code" type="xs:string" use="required"/>
530   <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
531   <xs:attribute name="comment" type="xs:string" use="optional"/>
532 </xs:complexType>
533
534 <xs:element name="Status" type="StatusType">
535   <xs:annotation>
536     <xs:documentation>
537       A standard Status type
538     </xs:documentation>
539   </xs:annotation>
540 </xs:element>
541
542
```

543 **Figure 3. <Status> Element and Type - Schema Fragment (from liberty-idwsf-utility-v2.0.xsd)**

544 In the two sections below, first the values of the code attribute of the <Status> element are discussed, followed by
 545 discussion of the various forms of <SASLResponse> messages and their semantics.

546 **4.6.2.1.1. Values for the code attribute of <status>**

547 If the value of code is:

- 548 • **"Continue"** — the server expects the client to craft and send a new `<SASLRequest>` message containing data
549 appropriate for whichever step the execution of the SASL mechanism is at.
- 550 • **"OK"** — the server considers the authentication exchange to have completed successfully.
- 551 The `<SASLResponse>` message will typically contain ID-WSF EPR(s) (i.e. `<wsa:EndpointReference>`
552 element(s)) containing credentials, as described below in [Section 5.3: Rules for Authentication Service Providers](#)
553 , enabling the client to interact further with this provider, for example to invoke another ID-WSF service such as
554 the Discovery Service.
- 555 Additionally, the `<SASLResponse>` message can convey ID-WSF EPRs for other providers.
- 556 See [Section 4.7: Sequencing of the Authentication Exchange](#) for the normative specification of the composition
557 of the `<SASLResponse>` message in this case. See also [Section 5.3: Rules for Authentication Service Providers](#) .
- 558 • **"Abort"** — the server is aborting the authentication exchange. It will not send any more messages on this message
559 thread.

560 4.6.2.1.2. Returning the Server's Selected SASL Mechanism

561 The server will choose one SASL mechanism from among the intersection of the list sent by the client and the server's
562 set of supported and willing-to-execute SASL mechanisms. It will return the name of this selected SASL mechanism
563 as the value for the `serverMechanism` attribute on the initial `<SASLResponse>` message. See [Example 5](#).

```
564  
565 <SASLResponse serverMechanism="DIGEST-MD5" >  
566   <Status code="Continue" />  
567   <Data>  
568     Q29ub3IqQ2FoaWxsIGNhc3VhbGx5IG1hbmdsZXMgcGFzc3dvcmRzCg==  
569   </Data>  
570 </SASLResponse>  
571
```

572 **Example 5. `<SASLResponse>` Indicating Server's Chosen SASL Mechanism**

573 If there is no intersection between the client-supplied list of SASL mechanisms and the set of supported, and willing-
574 to-execute, server-side SASL mechanisms, then the server will return a `<SASLResponse>` message with a `code`
575 attribute whose value is "Abort". See [Example 6](#), and also item #3 in [Section 4.7: Sequencing of the Authentication](#)
576 [Exchange](#) .

```
577  
578 <SASLResponse>  
579   <Status code="Abort" />  
580 </SASLResponse>  
581
```

582 **Example 6. `<SASLResponse>` Indicating a Server-side Abort**

583 4.7. Sequencing of the Authentication Exchange

584 The authentication exchange is sequenced as follows:

- 585 1. The authentication exchange **MUST** begin by the client sending the server a `<SASLRequest>` message. This
586 message:

- 587 • MUST contain a `mechanism` attribute whose value is a string containing one or more SASL mechanisms
588 the client supports and is prepared to negotiate (see [Section 4.6.1.2: Values for mechanism attribute of](#)
589 `<SASLRequest>`).
- 590 • MAY contain a `<Data>` element containing an initial response, specific to the cited SASL mechanism, if the
591 `mechanism` attribute contains only a single SASL mechanism. See section 5 of [\[RFC4422\]](#).
- 592 • MAY contain a `<samlp2:RequestedAuthnContext>` element.
- 593 • SHOULD contain an `authzID` attribute whose value is an identifier string for the Principal being authenti-
594 cated.
- 595 • MAY contain an `advisoryAuthnID` attribute whose value is an identifier asserted by the client to represent
596 the authentication identity being established by this authentication event.
- 597 2. If the server is prepared to execute, with this client, at least one of the SASL mechanism(s) cited by the client in
598 the previous step, then processing continues with step 4.
- 599 3. Otherwise, the server does not support, or is not prepared to negotiate, any of the SASL mechanisms cited by the
600 client. The server MUST respond to the client with a `<SASLResponse>` message containing:
- 601 • A `<Status>` element with a `code` attribute with a value of "**Abort**".
- 602 • No `<PasswordTransforms>` element.
- 603 • No `<Data>` element.
- 604 • No `<wsa:EndpointReference>` element.
- 605 • No `serverMechanism` attribute.
- 606 After this message is sent to the client, processing continues with step 7.
- 607 4. The server sends to the client a `<SASLResponse>` message.
- 608 If this message is the first `<SASLResponse>` sent to the client in this authentication exchange (as determined by
609 a particular authentication mechanism, see substep "**A. Continue**", below), this message:
- 610 • MUST contain a `serverMechanism` attribute whose value is a single SASL mechanism name, chosen by
611 the server from the list sent by the client.
- 612 • MAY contain a `<Data>` element containing a SASL mechanism-specific challenge.
- 613 • MAY contain a `<PasswordTransforms>` element. See [Section 8: Password Transformations: The](#)
614 `PasswordTransforms` Element for details on the client's subsequent obligations in this case.
- 615 • MUST contain a `<Status>` element with a `code` attribute whose value is given by either item **A**, or **B**, or **C**:

616 A. "**Continue**" — either the execution of the SASL mechanism is not complete or the authentication
617 exchange was successful but the server expects the client to authenticate again using a different
618 authentication mechanism; the server expects the client to process this message and respond.

619 If the server is indicating that the client should continue by authenticating with a different mechanism,
620 the server **MUST** specify the desired mechanism as the value for "serverMechanism". The authentication
621 mechanism specified **MUST** be taken from the list previously sent by the client in the prior authentication
622 exchange. The server **MAY** include a <Data> element (and <PasswordTransforms>) with content
623 appropriate for the new authentication mechanism.

624 If the reason for the server indicating that the client should continue is that the client pre-
625 sented invalid credentials, the server **SHOULD** include a second level status <Status
626 code="InvalidCredentials">. The server **MAY** also return a <Data> element (e.g. with a
627 new challenge according to the mechanism already established) and the client can respond according to
628 the mechanism. Processing continues with step 5.

629 B. "**OK**" — the server declares the authentication exchange has completed successfully.

630 In this case, this *final SASL response* message can contain, in addition to the items listed above,
631 <wsa:EndpointReference> element(s), containing requisite credentials. This is specified in
632 [Section 5.3: Rules for Authentication Service Providers](#) .

633 Processing continues with step 6.

634 C. "**Abort**" — the server declares the authentication exchange has completed unsuccessfully. For example,
635 the user may have supplied incorrect information, such as an incorrect password. See step 7, below, for
636 additional information.

637 In this case, this <SASLResponse> message **MUST NOT** contain any <wsa:EndpointReference>
638 element(s).

639 Processing continues with step 7.

640 Otherwise, this message:

641 • **MUST NOT** contain a serverMechanism attribute.

642 • **MAY** contain a <Data> element containing a SASL mechanism-specific challenge.

643 • **MUST NOT** contain a <PasswordTransforms> element.

644 • **MUST** contain a <Status> element with a code attribute whose value is given by either item A, or B, or C:

645 A. "**Continue**" — the execution of the SASL mechanism is not complete; the server expects the client to
646 process this message and respond. Processing continues with step 5.

647 B. "**OK**" — the server declares the authentication exchange has completed successfully.

648 In this case, this "final response" <SASLResponse> message can contain, in addition to the items
649 listed above, <wsa:EndpointReference> element(s) with requisite credentials. This is specified in
650 [Section 5.3: Rules for Authentication Service Providers](#) .

651 Processing continues with step 6.

- 652 C. **"Abort"** — the server declares the authentication process has completed unsuccessfully. For example,
653 the user may have supplied incorrect information, such as an incorrect password.
- 654 If the reason for the server aborting is that the client presented invalid credentials, the server SHOULD
655 include a second level status `<Status code="InvalidCredentials">`.
- 656 In this case, this `<SASLResponse>` message MUST NOT contain any `<wsa:EndpointReference>`
657 element(s).
- 658 Processing continues with step 7.
- 659 5. The client sends the server a `<SASLRequest>` message. This message:
- 660 • SHOULD contain a `mechanism` attribute set to the same value as sent by the server, as the value of the
661 `serverMechanism` attribute, in its first `<SASLResponse>` message (see [Section 4.6.2.1.2: Returning the](#)
662 [Server's Selected SASL Mechanism](#)).
- 663 **Note**
- 664 The client MAY, however, choose to abort the authentication exchange by setting the `mechanism` attribute
665 to either a "null" string, or to a mechanism name different than the one returned by the server in its first
666 `<SASLResponse>` message.
- 667 If the client chooses to abort, processing continues with step 8.
- 668 • SHOULD contain a `<Data>` element containing data specific to the cited SASL mechanism.
 - 669 • MUST NOT contain a `<samlp2:RequestedAuthnContext>` element.
- 670 Processing continues with steps 4 and 5 until the server signals success, failure, or aborts — or the client aborts
671 the exchange using the technique noted in the first bullet item, above, of this step.
- 672 6. The authentication exchange has completed successfully. The client is now authenticated in the server's view, and
673 the server may be authenticated in the client's view, depending upon the SASL mechanism employed. [Section 5.1:](#)
674 [Authentication Service: Conceptual Model](#) discusses what the next interaction steps between the client and
675 server are in the ID-WSF authentication service case.
- 676 7. The authentication exchange has completed unsuccessfully due to an exception on the server side. The client
677 SHOULD cease sending messages on this message thread.
- 678 The reasons for an authentication exchange failing are manifold. Often it is simply a case of the user having
679 supplied incorrect information, such as a password or passphrase. Or, there may have been a problem on the
680 server's part, such as an authentication database being unavailable or unreachable.
- 681 8. The client aborted the authentication exchange.

682 5. Authentication Service

683 The ID-WSF Authentication Service provides web service-based authentication facilities to Web Service Consumers
684 (WSCs). This service is built around the SASL-based ID-WSF Authentication Protocol as specified above in [Section 4](#).

685 This section first outlines the Authentication Service's conceptual model and then defines the service itself.

686 5.1. Conceptual Model

687 ID-WSF-based Web Service Providers (WSPs) may require requesters, AKA Web Service Consumers (WSCs), to
688 present security tokens in order to successfully interact (security token specifics, are specified in [\[LibertySecMech\]](#)).

689 A Discovery Service [\[LibertyDisco\]](#), which itself is just a WSP, is able to create security tokens authorizing WSCs to
690 interact with other WSPs, on whose behalf a Discovery Service has been configured to speak. Also, Discovery Service
691 instances might themselves be configured to require WSCs to present security tokens when making requests of them.

692 The ID-WSF Authentication Service addresses the above conundrum by providing the means for WSCs to prove their
693 identities—to authenticate—and obtain security tokens enabling further interactions with other services, at the same
694 provider, on whose behalf the Authentication Service instance is authorized to speak. These offered services may be,
695 for example, a Discovery Service or Single Sign-On Service. WSCs may then use these latter services to discover and
696 become capable of interacting with yet other services.

697 Note that although an Authentication Service itself does not require requesters to present security tokens in order to
698 interact with it, an Authentication Service may, in some situations, be configured to understand presented security
699 tokens and use them when applying policy.

700 5.1.1. Stipulating a Particular Authentication Context

701 In some situations, a WSC may need to stipulate some of the properties for an authentication exchange. A scenario
702 illustrating a use case of this is:

703 Suppose a Principal is wielding a Liberty-enabled user agent or device (LUAD) that is acting as
704 a WSC (i.e. a LUAD-WSC). The Principal authenticates with her bank, say, and authenticates
705 via the ID-WSF authentication service using some authentication mechanism, such as PLAIN
706 [\[SASLReg\]](#). At some point, the Principal wants to transfer a large sum of money to the Fund
707 for Poor Specification Editors (using some (fictitious) ID-SIS-based web service), and the bank's
708 system indicates to the LUAD-WSC that the Principal's present authentication is "inappropriate".
709 The bank's system also includes a `<RequestedAuthnContext>`.

710 Now, the LUAD-WSC "knows" that it needs to help the Principal reauthenticate—as her present
711 credentials aren't being honored for the financial transaction she wishes to carry out. So the
712 LUAD-WSC prompts the Principal for permission to reauthenticate her, and (assuming the answer
713 was "yes") initiates the ID-WSF Authentication Protocol with the appropriate authentication service
714 provider, and includes the supplied-by-the-bank `<RequestedAuthnContext>`. The authentication
715 service provider factors the requested authentication context into its selection of SASL mechanism
716 for the ensuing authentication exchange. And upon successful authentication, the Principal is able
717 to successfully make the funds transfer.

718 When initiating an authentication exchange, a WSC can stipulate some properties for the ensuing authentication event,
719 and thus the subsequently issued (if successful) credentials. It does this by including a `<RequestedAuthnContext>`
720 in the initial `<SASLRequest>`.

721 5.2. URI Declarations

722 The URI declarations for the ID-WSF Authentication Service are given below in [Table 2](#).

723

Table 2. Authentication Service URIs

Use	URI
Service Type	<i>urn:liberty:sa:2006-08</i>
SASLRequest wsa:Action	<i>urn:liberty:sa:2006-08:SASLRequest</i>
SASLResponse wsa:Action	<i>urn:liberty:sa:2006-08:SASLResponse</i>

724 **5.3. Rules for Authentication Service Providers**

725 Providers offering ID-WSF Authentication Services MUST adhere to the following rules:

726 1. Authentication Service Providers (AS Providers) MUST implement the ID-WSF Authentication Protocol, as
 727 defined in [Section 4: Authentication Protocol](#) . The Authentication Service Provider MUST play the role of the
 728 *authentication server*.

729 2. Upon successful completion of an authentication exchange the **first** ID-WSF EPR, as materialized as an
 730 `<wsa:EndpointReference>` element instance and contained in the *final SASL response*, SHOULD refer to
 731 services at the Authentication Service provider—i.e. at the "same provider"—that said AS Provider can offer to
 732 the Authentication Service consumer.

733 For example, Identity Providers may often also include an ID-WSF EPR for the Discovery Service of the Principal
 734 just authenticated, as well as ID-WSF EPRs for other offered services, such as an SSO Service.

735 **Note**

736 If the Authentication Service is invoked via a message whose indicated "framework version" [[LibertySOAPBinding](#)]
 737 is "2.0", then if the AS is returning ID-WSF EPRs for other services as noted above, then the AS SHOULD
 738 return ID-WSF EPRs for ID-WSFv2.0 services, rather than other versions of ID-WSF services.

739 See [Section 4.7: Sequencing of the Authentication Exchange](#) , Step 4.

740 The Provider MAY also include additional ID-WSF EPRs referring to services offered by other providers—i.e.
 741 providers other than the AS Provider.

742 3. Any included credentials SHOULD be useful for a reasonable time (note that credentials will be contained within
 743 the ID-WSF EPRs, as profiled in [[LibertyDisco](#)]). Even if the AS Consumer recently authenticated with the
 744 Authentication Service, i.e. an earlier issued credential for consumption by the AS Provider is still valid, the
 745 AS Provider SHOULD issue credential(s) that have later expiration times than the earlier issued credential(s).
 746 The AS Provider MAY choose to re-authenticate, using any of the available SASL mechanisms, or issue new
 747 credentials without an engaging in an authentication exchange. This can be accomplished by responding to the AS
 748 Consumer's initial SASL request with a final SASL response containing an ID-WSF EPR, itself containing the
 749 requisite credentials.

750 **Note**

751 Credentials containing `<saml2:AuthnStatement>(s)` should have their `<saml2:AuthnInstant>(s)` set to the
 752 time when the authentication event actually took place. See [[SAMLCore2](#)].

753 4. Additionally, if the first `<SASLRequest>` in an exchange contains a `<samlp2:RequestedAuthnContext>`
 754 element, then upon successful authentication, the Authentication Service MUST either: return credentials
 755 (embedded within returned ID-WSF EPR(s)) that satisfy the `<samlp2:RequestedAuthnContext>`, or, abort
 756 the authentication exchange. See [[SAMLCore2](#)] for a detailed description of the processing rules governing
 757 evaluation of the `<samlp2:RequestedAuthnContext>` element.

758 5. An Authentication Service instance SHOULD be deployed such that the security mechanism [[LibertySecMech](#)]:

759 `urn:liberty:security:2003-08:TLS:null`

760 can be used by the WSC.

761 **Note**

762 In practice this means that the Authentication Service should be exposed on an endpoint for which the URL
763 should have `https` as the protocol field.

764 6. An Authentication Service implementation SHOULD support the following SASL mechanisms [[SASLReg](#)]:
765 PLAIN, CRAM-MD5.

766 5.4. Rules for Authentication Service Consumers

767 WSCs implementing the client-side of the ID-WSF Authentication Protocol, and thus also known as *Authentication*
768 *Service Consumers* (AS Consumers), MUST adhere to the following rules:

769 1. AS Consumers MUST implement the ID-WSF Authentication Protocol, as defined in [Section 4: Authentication](#)
770 Protocol in the role of the client.

771 **Note**

772 The AS Consumer may include various SOAP header blocks, e.g. a `<wsse:Security>` element [[Liberty-](#)
773 [SecMech](#)] which can house a security token(s) obtained earlier from an Authentication Service or Discovery
774 Service [[LibertyDisco](#)]. In such a case, the Authentication Service SHOULD evaluate the presented security to-
775 ken(s) in combination with applicable policy, as a part of the overall authentication event. This provides a means,
776 for example, of "security token renewal".

777 2. In case the AS Consumer has not been provisioned with the `<disco:SecurityMechID>` for the Authentication
778 Service instance that it uses, the AS Consumer SHOULD assume that the required security mechanism is this
779 one:

780 `urn:liberty:security:2003-08:TLS:null`

781 **Note**

782 `<disco:SecurityMechID>` elements are contained within the `<disco:SecurityContext>` element(s),
783 themselves occurring within ID-WSF EPRs (profiled `<wsa:EndpointReference>`s) [[LibertyDisco](#)].

784 Only when the endpoint URL of the Authentication Service is prescribed to have `http` as the protocol MAY the
785 WSC presume a security mechanism of:

786 `urn:liberty:security:2003-08:null:null`

787 3. It is RECOMMENDED that the WSC support the password transformations specified in [Appendix B](#).

788 5.5. Authentication Service Interaction Example

789 [Example 7](#) through [Example 10](#) illustrate an example exchange between a LUAD-WSC and an ID-WSF Authentication
790 Service (AS). The AS includes information about the Discovery Service (DS) in its final response. Here the DS is
791 offered by the same provider.

```
792
793 <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
794   <s:Header>
795     ...
796   </s:Header>
797   <s:Body>
798     <SASLRequest mechanism="CRAM-MD5"
799       advisoryAuthnID="358408021451"
800       xmlns="urn:liberty:sa:2004-04" />
801   </s:Body>
802 </s:Envelope>
803
```

804 **Example 7. The WSC sends a <SASLRequest> on behalf of a Principal, asserting that the authentication identity is**
805 **"358408021451" and indicates it desire to use the "CRAM-MD5" SASL mechanism.**

```
806
807 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
808   <S:Header>
809     ...
810   </S:Header>
811   <S:Body>
812     <SASLResponse serverMechanism="CRAM-MD5"
813       xmlns="urn:liberty:sa:2004-04">
814       <Status code="continue"/>
815       <Data>
816         ...a CRAM-MD5 challenge here...
817       </Data>
818     </SASLResponse>
819   </s:Body>
820 </s:Envelope>
821
```

822 **Example 8. The AS replies, agreeing to use CRAM-MD5, and issues a CRAM-MD5 challenge.**

```
823
824 <S:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
825   <s:Header>
826     ...
827   </s:Header>
828   <s:Body>
829     <SASLRequest mechanism="CRAM-MD5"
830       xmlns="urn:liberty:sa:2004-04">
831       <Data>
832         ...some CRAM-MD5 response here...
833       <Data>
834     </SASLRequest>
835   </s:Body>
836 </s:Envelope>
837
```

838 **Example 9. The WSC responds with a CRAM-MD5 response.**

```

839
840 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
841   <S:Header>
842     ...
843   </S:Header>
844   <S:Body id="msgBody">
845
846     <sa:SASLResponse xmlns:sa="urn:liberty:sa:2004-04"
847       xmlns:disco="urn:liberty:disco:2003-08">
848       <Status code="sa:OK"/>
849
850       <wsa:EndpointReference>
851
852         <wsa:Address>
853           http://tg2.example.com:8080/tfs-soap/IdPDiscoveryService
854         </wsa:Address>
855
856         <wsa:Metadata>
857
858           <disco:ServiceType>urn:liberty:disco:2003-08</disco:ServiceType>
859
860           <disco:ProviderID>http://tg2.example.com:8080/tfs</disco:ProviderID>
861
862           <ds:SecurityContext>
863
864             <disco:SecurityMechID>
865               urn:liberty:security:2005-02:null:Bearer
866             </disco:SecurityMechID>
867
868             <sec:Token>
869               <saml2:Assertion
870                 ID="ilb42508103cab657f34e5ef189f28ea10dd86926 "
871                 Version="2.0"
872                 IssueInstant="2004-02-03T22:12:33Z">
873                 <Issuer>
874                   http://tg2.trustgenix.com:8080/tfs
875                 </Issuer>
876                 ....
877                 ....
878               </saml2:Assertion>
879             </sec:Token>
880
881           </ds:SecurityContext>
882
883         </wsa:Metadata>
884
885       </wsa:EndpointReference>
886
887     </sa:SASLResponse>
888
889   </S:Body>
890 </S:Envelope>
891

```

892 **Example 10.** The AS replies with its "final" <SASLResponse> message, which includes credentials with which the WSC
 893 may subsequently use to invoke a DS.

894 6. Single Sign-On Service

895 The ID-WSF Single Sign-On Service (SSO Service, or SSOS) provides requesters with an ID-WSF-based means
896 to obtain *SAML 2.0 authentication assertions* enabling them to interact with *SAML 2.0 Service Providers* (SPs)
897 [SAMLCore2] as well as other services that accept SAML 2.0 assertions as security tokens, such as web services
898 (including ID-WSF WSPs). The SSOS is based on a pair of profiles of the SAML 2.0 Authentication Request protocol
899 [SAMLCore2], one of which is a refinement of the SAML 2.0 Enhanced Client or Proxy SSO profile [SAMLProf2].

900 This section first outlines the ID-WSF SSO Service's conceptual model and then defines the SSO Service in terms of
901 the SAML profiles it supports.

902 6.1. Conceptual Model

903 In the Liberty architecture, it is conceivable for any concrete *system entity* to don any architectural *role* that it is
904 physically capable of bearing. For example, a Liberty *Service Provider* (SP) is essentially just a SAML 2.0 SSO-
905 enabled website. Such Service Providers can also be simultaneously cast into WSC and WSP roles.

906 Similarly, *user agents* in the Liberty architecture range from standard web browsers, to modestly Liberty-enabled
907 browsers (ECPs), to arbitrarily complex SOAP-based clients. These latter user agents, termed *Liberty-enabled User*
908 *Agents or Devices* (LUADs) will conceivably be dynamically cast into the full range of Liberty architectural roles;
909 they will be called upon to be a browser one moment, and a WSC the next, and even a WSP at times.

910 As noted in Section 5, a (LUAD-)WSC that needs to obtain security tokens in order to interact with a Discovery Service
911 (and subsequently other ID-WSF services) can utilize an ID-WSF Authentication Service to obtain the requisite
912 security tokens. However, not all useful services (SOAP-based or otherwise) that accept SAML security tokens will
913 be registered with a Discovery Service. Furthermore, SAML 2.0 SSO-enabled web sites often rely on the ability to
914 issue requests for authentication directly to less capable clients and expect them to relay the request and subsequent
915 response. LUADs thus need a way to participate in that exchange.

916 Another class of use cases involves calls by one principal (or a WSC acting on behalf of a principal) to invoke services
917 belonging to another principal. These so-called cross-principal invocations often require a WSC to utilize the invoking
918 principal's Single Sign-On Service to obtain security tokens for the target principal's Discovery Service or other ID-
919 WSF services.

920 The ID-WSF Single Sign-On Service addresses these use cases with profiles of the SAML 2.0 Authentication Request
921 protocol [SAMLCore2]. Two distinct, but similar, profiles are defined in order to address differences that arise in
922 the content of security tokens, and protocol processing behavior that is specific to SAML 2.0 SSO SPs. The profile
923 addressing these SPs is a refinement or specialization of the existing SAML 2.0 Enhanced Client/Proxy SSO Profile
924 [SAMLProf2]. A SAML 2.0 SP can treat a LUAD in the same way as any other enhanced client. A second, more
925 generic, profile permits a LUAD to obtain SAML assertions useful in accessing other kinds of services, including use
926 cases defined in the future.

927 In both profiles, requesters authenticate to the SSOS using ID-WSF security mechanisms, making the SSOS itself
928 an ID-WSF service. A LUAD wishing to interact with the SSOS can use the Authentication Service at an Identity
929 Provider (IdP) to obtain security tokens that enable it to invoke the SSOS at that IdP in order to obtain additional
930 security tokens to convey to SAML 2.0 SPs or other SAML-enabled services.

931 In fact, if a LUAD successfully authenticates with an IdP via the IdP's Authentication Service Section 5, the IdP
932 can ensure that the LUAD will have in its possession an ID-WSF EPR (a profiled `<wsa:EndpointReference>`;
933 [LibertyDisco]), containing any necessary credentials, for the ID-WSF Single Sign-On Service at the same IdP,
934 simplifying the process of invoking the SSOS. Additionally, the IdP can, at the same time, ensure that the LUAD
935 possesses an ID-WSF EPR containing any necessary credentials for the Discovery Service (DS) of the Principal
936 wielding the LUAD, thus enabling the LUAD to simultaneously utilize SAML and ID-WSF-based services on behalf
937 of the Principal based on one sign-on interaction, from the Principal's perspective.

938 6.2. Single Sign-On Service URIs

939

Table 3. Single Sign-On Service URIs

Use	URI
Service Type	<i>urn:liberty:ssos:2006-08</i>
AuthnRequest wsa:Action	<i>urn:liberty:ssos:2006-08:AuthnRequest</i>
Response wsa:Action	<i>urn:liberty:ssos:2006-08:Response</i>

940 **6.3. ID-WSF Enhanced Client or Proxy SSO Profile**

941 The SAML 2.0 Enhanced Client or Proxy SSO Profile [SAMLProf2] enables SSO to web sites by SAML-aware
 942 clients, but leaves the authentication of the client to the IdP out of scope. This profile is a refinement that adds the
 943 ID-WSF SOAP Binding [LibertySOAPBinding] and Security Mechanisms [LibertySecMech] specifications to the
 944 communication with the IdP, enabling a LUAD to participate in SSO to SAML 2.0-enabled web sites.

945 **6.3.1. Profile Overview**

946 As introduced above, this profile is simply a constrained version of the interactions specified in [SAMLProf2].
 947 Specifically, it adds additional requirements to steps 4-6 of the ECP Profile, which involve the interactions between
 948 the IdP and the client. In all other respects, all processing rules defined by the base profile, and in turn the underlying
 949 SAML 2.0 Browser SSO Profile are observed. In particular, note that the content of the SAML protocol messages and
 950 assertion(s) used in this constrained version are entirely unchanged from [SAMLProf2].

951 **6.3.2. Profile Description**

952 The following sections provide detailed definitions of the individual steps. Except where noted, the steps and
 953 processing rules are as specified in the ECP Profile [SAMLProf2].

954 1. LUAD issues HTTP Request to Service Provider

955 Step 1 is identical to step 1 of the ECP Profile, but MAY be omitted in cases in which the LUAD wishes to
 956 construct the request to the IdP itself and possesses sufficient knowledge of the SP to do so.

957 2. Service Provider issues <samlp2:AuthnRequest> to Identity Provider via LUAD

958 Step 2 is identical to step 2 of the ECP Profile, but note that the <samlp2:AuthnRequest> message MAY be
 959 constructed independently by the LUAD rather than obtained from the SP. From the perspective of the SP, the
 960 eventual response in step 7 will be treated as unsolicited.

961 3. LUAD Determines Identity Provider

962 Step 3, out of scope in the ECP Profile, is similarly out of scope here.

963 4. LUAD forwards <samlp2:AuthnRequest> to Identity Provider

964 In step 4, the <samlp2:AuthnRequest> message is sent to the selected IdP's ID-WSF Single Sign-On Service
 965 endpoint using the Liberty SOAP binding [LibertySOAPBinding]. This message MUST be authenticated using a
 966 security mechanism defined by [LibertySecMech].

967 When communicating with the Identity Provider, the client MUST adhere to the Liberty SOAP binding as
 968 specified in [LibertySOAPBinding]; in case of conflict with the SOAP binding as specified in [SAMLBind2]
 969 the Liberty SOAP Binding shall take precedence.

970 **Note**

971 The client MAY (and generally will) include various other header blocks, e.g. a `<wsse:Security>` header
972 block [[LibertySecMech](#)] [[wss-sms](#)]. Such a header block could contain a security token obtained from the ID-
973 WSF Authentication Service.

974 Note that the `<samlp2:AuthnRequest>` message may also be signed by the SP (or the LUAD if constructed
975 by it). In this and other respects, the message rules specified in the SAML 2.0 Browser SSO profile in Section
976 4.1.4.1 of [[SAMLProf2](#)] MUST be observed.

977 5. Identity Provider Identifies Principal

978 In step 5, the ID-WSF peer-entity authentication mechanism used by the LUAD in step 4 MUST be used to
979 identify the Principal.

980 6. Identity Provider issues `<samlp2:Response>` message to Service Provider via LUAD

981 Step 6 is identical to step 6 of the ECP Profile, except for the use of the Liberty SOAP Binding during the
982 exchange. The SSOS SHOULD NOT respond in step 6 with any content other than SOAP. For example, the
983 MIME type of the HTTP response must be set according to [[LibertySOAPBinding](#)].

984 **Note**

985 This is different from the SAML 2.0 ECP profile [[SAMLProf2](#)] in which an IdP is permitted to respond with any
986 content acceptable to the requester during the authentication process.

987 The SSOS MAY take advantage of various optional header blocks defined in [[LibertySOAPBinding](#)]. For
988 example, instead of attempting to establish a local session via an HTTP cookie, the SSOS may include a
989 `<disco:SecurityContext>` element in an `<sb:EndpointUpdate>` header block. The requester must of
990 course understand such header blocks.

991 7. LUAD forwards `<samlp2:Response>` to Service Provider

992 Step 7 is identical to step 7 of the ECP Profile. When the client receives the `<samlp2:Response>`
993 message from the IdP, it MUST NOT forward it to any location other than that specified in the
994 `AssertionConsumerServiceURL` attribute contained in the mandatory `<ecp:Response>` header block
995 received from the IdP.

996 Note however that in the case that the LUAD initiated the profile by constructing the `<samlp2:AuthnRequest>`
997 message in step 2, then there is no explicit comparison to be made against the `AssertionConsumerServiceURL`
998 attribute in the ECP Response header block.

999 8. Service Provider Grants or Denies Access to Principal

1000 Step 8 is identical to step 8 of the ECP profile.

1001 **6.4. ID-WSF SAML Token Service Profile**

1002 The SAML Token Service Profile is an ID-WSF-based profile of the SAML 2.0 Authentication Request protocol
1003 [[SAMLCore2](#)] that permits a requester to obtain SAML assertions for use by one or more relying parties. Relying
1004 parties might be ID-WSF-based web services, generically defined web services, or other application services that
1005 support SAML. The profile is a direct exchange between the requester and the IdP offering the token service using
1006 the ID-WSF SOAP Binding [[LibertySOAPBinding](#)] and is authenticated using the ID-WSF Security Mechanisms
1007 specification [[LibertySecMech](#)].

1008 **6.4.1. Profile Overview**

1009 As introduced above, this profile uses the SAML 2.0 Authentication Request protocol [[SAMLCore2](#)] to enable an IdP
1010 to offer a relatively unconstrained capability to issue SAML assertions containing authentication and other information
1011 as security tokens for use by the requester with specified relying parties. Unlike the SSO-oriented profiles defined in

1012 [SAMLProf2] and the previous section, this profile directly involves only two parties: the requester (e.g. a LUAD) and
1013 the IdP. The client in this profile is the actual requester, rather than an intermediary between the IdP and the eventual
1014 relying party.

1015 Operationally, another difference between this profile and the SSO profiles relates to the content of the SAML
1016 assertions that can be issued. Other than a few basic assumptions, the IdP is generally expected to have sufficient
1017 knowledge of the relying parties identified by the requester so as to support the issuance of appropriately constructed
1018 assertions supporting those parties' requirements. The means by which it can obtain such knowledge are out of scope,
1019 and could include the out of band exchange of policy information, direct configuration, or it may rely on the requester
1020 to indicate to it what kinds of information to include.

1021 This profile is a combination of the SAML Authentication Request protocol and the ID-WSF SOAP Binding
1022 [LibertySOAPBinding] and Security Mechanisms [LibertySecMech] specifications, along with a few guidelines on
1023 the use of the <samlp2:AuthnRequest> message.

1024 6.4.2. Profile Description

1025 The following sections provide detailed definitions of the individual steps.

1026 1. Requester issues <samlp2:AuthnRequest> to Identity Provider

1027 In step 1, the requester sends its <samlp2:AuthnRequest> message to the selected IdP's ID-WSF Single
1028 Sign-On Service endpoint using the Liberty SOAP binding [LibertySOAPBinding]. This message MUST be
1029 authenticated using a security mechanism defined by [LibertySecMech].

1030 When communicating with the Identity Provider, the client MUST adhere to the Liberty SOAP binding as
1031 specified in [LibertySOAPBinding]; in case of conflict with the SOAP binding as specified in [SAMLBind2]
1032 the Liberty SOAP Binding shall take precedence.

1033 **Note**

1034 The client MAY (and generally will) include various other header blocks, e.g. a <wsse:Security> header
1035 block [LibertySecMech] [wss-sms]. Such a header block could contain a security token obtained from the ID-
1036 WSF Authentication Service.

1037 2. Identity Provider Identifies Principal

1038 In step 2, the ID-WSF peer-entity authentication mechanism used by the requester in step 1 MUST be used to
1039 identify the requesting Principal.

1040 3. Identity Provider issues <samlp2:Response> message to Requester

1041 In step 3, the IdP returns a <samlp2:Response> message to the requester containing the status and any SAML
1042 assertion(s) issued as a result of the request. The SSOS SHOULD NOT respond with any content other than
1043 SOAP. For example, the MIME type of the HTTP response must be set according to [LibertySOAPBinding].

1044 The SSOS MAY take advantage of various optional header blocks defined in [LibertySOAPBinding]. For
1045 example, instead of attempting to establish a local session via an HTTP cookie, the SSOS may include a
1046 <disco:SecurityContext> element in an <sb:EndpointUpdate> header block. The requester must of
1047 course understand such header blocks.

1048 **6.4.3. Use of SAML 2.0 Authentication Request Protocol**

1049 This profile is based on the SAML 2.0 Authentication Request protocol defined in [SAMLCore2]. In the nomenclature
1050 of actors enumerated in Section 3.4 of that document, the requester is the SAML requester, presenter, and the attesting
1051 entity, and is generally the requested subject. Relying parties may be identified in the request but are not a party to
1052 the profile. There may be additional attesting entities and relying parties at the discretion of the identity provider (see
1053 below).

1054 **6.4.3.1. <samlp2:AuthnRequest> Usage**

1055 Except as described below, a requester MAY include any message content described in [SAMLCore2], Section 3.4.1.
1056 All processing rules are as defined in [SAMLCore2].

1057 The <saml2:Issuer> element MUST NOT be present. This avoids duplication or conflict with the mandatory
1058 information already available from the ID-WSF SOAP Binding.

1059 If the IdP cannot or will not satisfy the request, it MUST respond with a <samlp2:Response> message containing
1060 an appropriate error status code or codes.

1061 The ProtocolBinding attribute MUST be included and MUST be set to `http://www.w3.org/2005/08/addressing/anonymous`
1062 to indicate the response is to be returned directly to the requester. This value clearly distinguishes this pro-
1063 file's use of the protocol from that of the SAML 2.0 SSO profiles. The request MUST NOT contain the
1064 AssertionConsumerServiceURL or AssertionConsumerServiceIndex attributes.

1065 If no <saml2:Subject> element is included, the invocation identity associated with the request is implied to be the
1066 requested subject. The requester MAY explicitly include a <saml2:Subject> element in the request that names
1067 the actual Principal about which it wishes to receive an assertion. If the IdP does not recognize the requester as that
1068 Principal (or an entity allowed to attest to it), then it MUST respond with a <samlp2:Response> message containing
1069 an error status and no assertions.

1070 In most cases, the identifier to return for the requested subject can be determined based on the identity of the relying
1071 party or parties. If this is not sufficient (for example if the relying party is to be treated as a member of an affiliation
1072 of providers), then the <samlp2:NameIDPolicy> element can be used to indicate this using the SPNameQualifier
1073 attribute. (Note that this precludes the identification of multiple relying parties in a single request.)

1074 If the requester wishes to permit the IdP to establish a new identifier for the Principal if none exists, it MUST include
1075 a <samlp2:NameIDPolicy> element with the AllowCreate attribute set to "true". Otherwise, only a principal
1076 for whom the IdP has previously established an identifier usable by the relying party or parties can be authenticated
1077 successfully.

1078 The requester MAY include one or more <saml2:SubjectConfirmation> elements in the request to specify
1079 attestation mechanisms to be attached to the resulting assertion(s). Usually this is done to translate ID-WSF security
1080 mechanism requirements into the corresponding SAML confirmation methods that will be needed to satisfy the
1081 relying party's security policy. Refer to [LibertySecMech] for a detailed mapping of security mechanisms to SAML
1082 confirmation methods.

1083 The requester MAY include a <saml2:Conditions> element in the request. The IdP is NOT obligated to honor the
1084 requested conditions, but SHOULD return an error if it cannot do so.

1085 The requester MAY include an <saml2:AudienceRestriction> element in the request to enumerate one or more
1086 relying parties by means of a <saml2:Audience> element containing a unique identifier for the relying party. The
1087 IdP can utilize whatever knowledge is at its disposal to determine the appropriate content to place in the resulting
1088 assertion(s), as well as how many assertions to issue, and whether the use of XML encryption is required.

1089 The request MUST be signed or otherwise integrity protected by the binding or transport layer.

1090 **6.4.3.2. <samlp2:Response > Usage**

1091 If the IdP wishes to return an error, it **MUST NOT** include any assertions in the <samlp2:Response> message.
1092 Otherwise the <samlp2:Response> element **MUST** conform to the following:

- 1093 • The <saml2:Issuer> element **MAY** be omitted, but if present **MUST** contain the unique
1094 identifier of the issuing IdP; the Format attribute **MUST** be omitted or have a value of
1095 urn:oasis:names:tc:SAML:2.0:nameid-format:entity
- 1096 • It **MUST** contain at least one <saml2:Assertion>. Each assertion's <saml2:Issuer> element **MUST**
1097 contain the unique identifier of the issuing IdP; the Format attribute **MUST** be omitted or have a value of
1098 urn:oasis:names:tc:SAML:2.0:nameid-format:entity
- 1099 • Each assertion returned **MUST** be signed.
- 1100 • The set of one or more assertions **MUST** contain at least one <saml2:AuthnStatement> that reflects the
1101 authentication of the subject to the IdP.
- 1102 • Confirmation methods and additional statements **MAY** be included in the assertion(s) at the discretion of the IdP.
1103 In particular, <saml2:AttributeStatement> elements **MAY** be included.
- 1104 • The assertions **SHOULD** contain an <saml2:AudienceRestriction> condition element containing the unique
1105 identifier of the intended relying party or parties within the included <saml2:Audience> elements.
- 1106 • Other conditions (and other <saml2:Audience> elements) **MAY** be included as requested or at the discretion
1107 of the IdP. Of course, all such conditions **MUST** be understood and accepted by the relying party in order for the
1108 assertion to be considered valid.

1109 **6.5. Use of Metadata**

1110 An IdP that offers an ID-WSF Single Sign-On Service supporting either of the profiles above **SHOULD** ad-
1111 vertise support for this capability in its metadata [SAMLMeta2]. To accomplish this, it **MUST** include a
1112 <md:SingleSignOnService> element in its metadata with a Binding attribute of urn:liberty:sb:2006-08.

1113 The <md:IDPSSODescriptor> element's WantAuthnRequestsSigned attribute **MAY** be used by an IdP to
1114 document a requirement that requests be signed (as opposed to protected only at the transport layer).

1115 **6.6. Inclusion of ID-WSF Endpoint References**

1116 Both SSOS profiles support the inclusion of arbitrary content in the assertions returned. Specifically, the SSOS **MAY**
1117 include ID-WSF EPRs, encoded as SAML attributes, for the associated Principal's Discovery Service or other ID-
1118 WSF-based services. These EPRs **MAY** contain additional security tokens or **MAY** refer to the containing assertion if
1119 appropriate.

1120 7. Identity Mapping Service

1121 The ID-WSF Identity Mapping Service enables a requester to obtain one or more "identity tokens". As defined in
1122 [\[LibertySecMech\]](#), identity tokens can be used to refer to principals in a privacy-preserving manner. The Identity
1123 Mapping Service is an ID-WSF web service that translates references to a principal into alternative formats or identifier
1124 namespaces. It is a generalization of the Name Identifier Mapping protocol defined in [\[SAMLCore2\]](#).

1125 This section first outlines the service's conceptual model and then defines the protocol and message elements.

1126 7.1. Conceptual Model

1127 The ID-WSF Identity Mapping Service allows a requester in possession of one or more identity tokens to translate,
1128 update, or refresh them using the protocol defined here. An example employer of this service is the ID-WSF People
1129 Service [\[LibertyPeopleService\]](#). A principal may also act on behalf of itself (or in conjunction with a WSC) to obtain
1130 an identity token representing that principal for use in subsequent web service invocations.

1131 An identity token can take a variety of forms, including many SAML-based representations such as SAML assertions
1132 or SAML identifier fragments (encrypted or plaintext) such as may appear in the subject of SAML assertions (e.g.
1133 elements such as `<saml2:NameID>` or `<saml2:EncryptedID>`). A security token, such as a SAML authentication
1134 assertion can also serve as an identity token. Note that when evaluating the validity of an identity token in SAML
1135 assertion form, constraints may be imposed on its use by the issuer of the assertion.

1136 SAML assertions used as identity tokens can also be used to communicate ID-WSF EPR and credential information,
1137 such as for the associated Principal's Discovery Service or other ID-WSF-based services.

1138 Conceptually, the mapping protocol is a translation or exchange of one or more inputs for corresponding outputs.
1139 Each input consists of an identity token and a policy specifying the identity token to return. The security token of the
1140 invoking identity can also be referenced as the input token. The output is the requested identity token, the exact form
1141 of which may be up to the mapping service to establish.

1142 7.2. Schema Declarations

1143 The XML schema [\[Schema1-2\]](#) normatively defined in this section is constituted in the XML Schema file:
1144 `liberty-idwsf-idmapping-svc-v2.0.xsd`, entitled " [Liberty ID-WSF Identity Mapping Service XSD v2.0](#) "
1145 (see [Appendix D](#)).

1146 Additionally, [Liberty ID-WSF Identity Mapping Service XSD v2.0](#) imports items from `liberty-idwsf-utility-v2.0.xsd`
1147 (see [Appendix E: Liberty ID-WSF Utility XSD v2.0](#)), and also from `liberty-idwsf-security-mechanisms-v2.0.xsd`
1148 (see [\[LibertySecMech\]](#)).

1149 7.3. SOAP Binding

1150 The Identity Mapping Service is an ID-WSF service; as such, the messages defined in [Section 7.4](#) constitute *ordinary*
1151 *ID-* messages* as defined in [\[LibertySOAPBinding\]](#). They are intended to be bound to the [\[SOAPv1.1\]](#) protocol by
1152 mapping them directly into the `<s:Body>` element of the `<s:Envelope>` element comprising a SOAP message.

1153 [\[LibertySOAPBinding\]](#) normatively specifies this binding, as well as various required and optional SOAP header
1154 blocks usable with this protocol.

1155 7.3.1. Identity Mapping Service URIs

1156 **Table 4. Identity Mapping Service URIs**

Use	URI
Service Type	<i>urn:liberty:ims:2006-08</i>
IdentityMappingRequest wsa:Action	<i>urn:liberty:ims:2006-08:IdentityMappingRequest</i>
IdentityMappingResponse wsa:Action	<i>urn:liberty:ims:2006-08:IdentityMappingResponse</i>

1157 **7.4. Protocol Messages and Usage**

1158 The following request and response messages make up the Identity Mapping Service protocol:

1159 **7.4.1. Element <IdentityMappingRequest>**

1160 The <IdentityMappingRequest> element is of complex type **IdentityMappingRequestType**, and is defined in
 1161 [Figure 4](#) and in [Liberty ID-WSF Identity Mapping Service XSD v2.0](#) . This type contains the following attributes and
 1162 elements:

1163 • **Any Attributes** [Optional]

1164 Zero or more extension attributes qualified by an XML namespace other than the Identity Mapping Service
 1165 namespace.

1166 • **<MappingInput>** [One or More]

1167 One or more elements specifying the principals to return identity tokens for, and the policies describing the contents
 1168 of those tokens.

```

1169 <xs:element name="IdentityMappingRequest" type="IdentityMappingRequestType" />
1170 <xs:complexType name="IdentityMappingRequestType">
1171   <xs:sequence>
1172     <xs:element ref="MappingInput" maxOccurs="unbounded" />
1173   </xs:sequence>
1174   <xs:anyAttribute namespace="##other" processContents="lax" />
1175 </xs:complexType>
    
```

1177 **Figure 4. Element <IdentityMappingRequest> Schema Fragment**

1178 **7.4.1.1. Element <MappingInput>**

1179 The <MappingInput> element is of complex type **MappingInputType**, and is defined in [Figure 5](#) and in [Liberty](#)
 1180 [ID-WSF Identity Mapping Service XSD v2.0](#) . This type contains the following attributes and elements:

1181 • **reqID** [Optional]

1182 Uniquely identifies a <MappingInput> element within a request. Used by the responder to correlate the
 1183 <MappingOutput> elements that it returns to their corresponding inputs.

1184 • **<sec:TokenPolicy>** [Optional]

1185 A container for information specifying the characteristics of the identity token the requester wants returned to it.

1186 • **<sec:Token>** [Required]

1187 A container for the identity token (or token reference) that specifies the principal for whom to return a new identity
 1188 token.

1189 **Note**

1190 Notwithstanding the schema definition below that declares the `<sec:Token>` element to be optional, a
1191 `<sec:Token>` element **MUST** be present in the `<MappingInput>` element. The looser schema definition is
1192 designed to enable extension of the type by other specifications for which a mandatory `<sec:Token>` element
1193 may not be appropriate.

```
1194  
1195 <xs:element name="MappingInput" type="MappingInputType"/>  
1196 <xs:complexType name="MappingInputType">  
1197   <xs:sequence>  
1198     <xs:element ref="sec:TokenPolicy" minOccurs="0"/>  
1199     <xs:element ref="sec:Token" minOccurs="0"/>  
1200   </xs:sequence>  
1201   <xs:attribute name="reqID" type="lu:IDType" use="optional"/>  
1202 </xs:complexType>
```

1203 **Figure 5. Element `<MappingInput>` Schema Fragment**1204 **7.4.1.2. Request Usage**

1205 An `<IdentityMappingRequest>` consists of one or more `<MappingInput>` elements. If multiple
1206 `<MappingInput>` elements are included in a request, then each element **MUST** contain a `reqID` attribute so
1207 that the response contents can be correlated to them.

1208 Each input consists of an identity (in the form of a `<sec:Token>`) and an optional `<sec:TokenPolicy>`.

1209 The input identity token describes the principal for whom the requester desires a new identity token. This **MAY** be a
1210 reference to a token elsewhere in the message or in some other location.

1211 The input token policy describes the nature of the identity token to be returned, generally focusing on the nature of
1212 the identifier. Often a principal will possess many alternate identifiers of different formats or scoped to different usage
1213 contexts, such as a particular SP or affiliation. The policy allows the requester to translate from the input token to
1214 some other form.

1215 If no token policy is supplied, then the resulting output token **SHOULD** have the same general characteristics as the
1216 input token, save perhaps for associated information such as its lifetime. This might be used to renew a token, for
1217 example.

1218 As identity tokens come in a variety of forms, so too the form of the input policy can vary. In the specific case of a
1219 SAML-based identity token, a `<samlp2:NameIDPolicy>` **SHOULD** be used, as defined in [[SAMLCore2](#)].

1220 The token policy **SHOULD** identify the entity for whom the identity token is being created, if other than the requester.
1221 If this cannot be otherwise inferred from the policy, the `issueTo` attribute **SHOULD** be used to identify this entity.
1222 When the `<samlp2:NameIDPolicy>` is used, the `SPNameQualifier` attribute will often supply this information, at
1223 least for persistent identifiers in typical use cases, making use of the `issueTo` attribute redundant.

1224 **7.4.2. Element `<IdentityMappingResponse>`**

1225 The `<IdentityMappingResponse>` element is of complex type **IdentityMappingResponseType**, and is defined in
1226 [Figure 6](#) and in [Liberty ID-WSF Identity Mapping Service XSD v2.0](#). This type contains the following attributes and
1227 elements:

- 1228 • **Any Attributes** [Optional]
1229 Zero or more extension attributes qualified by an XML namespace other than the Identity Mapping Service
1230 namespace.
- 1231 • **<lu:Status>** [Required]
1232 The status of the request. This element is defined in [Liberty ID-WSF Utility XSD v2.0](#).
- 1233 • **<MappingOutput>** [Zero or More]
1234 Zero or more elements containing the identity tokens returned.
- 1235
1236

```
<xs:element name="IdentityMappingResponse" type="IdentityMappingResponseType"/>
```


1237

```
<xs:complexType name="IdentityMappingResponseType">
```


1238

```
  <xs:sequence>
```


1239

```
    <xs:element ref="lu:Status"/>
```


1240

```
    <xs:element ref="MappingOutput" minOccurs="0" maxOccurs="unbounded"/>
```


1241

```
  </xs:sequence>
```


1242

```
  <xs:anyAttribute namespace="##other" processContents="lax"/>
```


1243

```
</xs:complexType>
```

1244 **Figure 6. Element <IdentityMappingResponse> Schema Fragment**

1245 7.4.2.1. Element <MappingOutput>

1246 The <MappingOutput> element is of complex type **MappingOutputType**, and is defined in [Figure 7](#) and in [Liberty](#)
1247 [ID-WSF Identity Mapping Service XSD v2.0](#). This type contains the following attributes and elements:

- 1248 • **reqRef** [Optional]
1249 Uniquely identifies a <MappingInput> element within the corresponding request. Used to correlate
1250 <MappingOutput> elements to their corresponding inputs.
- 1251 • **<sec:Token>** [Required]
1252 A container for the identity token (or token reference) returned by the responder.
- 1253
1254

```
<xs:element name="MappingOutput" type="MappingOutputType"/>
```


1255

```
<xs:complexType name="MappingOutputType">
```


1256

```
  <xs:sequence>
```


1257

```
    <xs:element ref="sec:Token"/>
```


1258

```
  </xs:sequence>
```


1259

```
  <xs:attribute name="reqRef" type="lu:IDReferenceType" use="optional"/>
```


1260

```
</xs:complexType>
```

1261 **Figure 7. Element <MappingOutput> Schema Fragment**

1262 7.4.2.2. Response Usage

1263 An <IdentityMappingResponse> consists of a status element and zero or more <MappingOutput> elements. If
1264 multiple <MappingInput> elements were included in a request, then each output element **MUST** contain a `reqRef`
1265 attribute matching it to the corresponding input element.

1266 Each output element consists of an identity token (in the form of a <sec:Token>).

1267 Any tokens returned MUST be constructed in accordance with the policy supplied in the input. A specific exception
1268 to this requirement is that any `validUntil` attribute specified by the requester MAY be ignored. If no policy was
1269 specified, the identity token to return is presumed to be of the same nature as the identity token used as input.

1270 The responder MUST take appropriate steps to ensure the privacy of the principal by encrypting the resulting identity
1271 information such that only the principal and parties known to be privy to the information can read it.

1272 If each input element is satisfied in the resulting response, then the responder MUST return a top-level `<lu:Status>`
1273 code of "OK".

1274 If at least one, but not all, of the resulting inputs cannot be satisfied, then the responder MUST return a top-level
1275 `<lu:Status>` code of "Partial". It MAY return nested `<lu:Status>` elements reflecting the specific result of the
1276 failed inputs.

1277 If none of the resulting inputs can be satisfied, then the responder MUST return a top-level `<lu:Status>` code of
1278 "Failed". It MAY return nested `<lu:Status>` elements reflecting the specific result of the failed inputs.

1279 When multiple inputs are present, any nested `<lu:Status>` elements MUST contain a `ref` attribute equal to the
1280 associated `<MappingInput>`'s `reqID` attribute.

1281 7.4.2.3. Second-Level Status Codes

1282 The following second-level codes are defined to represent common error conditions that may arise. Others may be
1283 defined by implementations as required.

1284 • "UnknownPrincipal" — the input token did not match a principal known to the service

1285 • "BadInput" — the input token or policy was malformed or not understood

1286 • "Denied" — the requested token translation was a violation of user or system policy

1287 7.5. SAML Identity Tokens

1288 As described in [[LibertySecMech](#)], identity tokens can be expressed in many ways. Even in the specific case of SAML,
1289 many different formulations are possible, depending on the requirements. This section outlines a few common ways of
1290 expressing identification using SAML with different security and privacy characteristics. The identity mapping service
1291 is responsible for selecting an appropriate choice based on the requester, input policy, and the expected purpose. It is
1292 outside the scope of this specification how such purposes are to be understood.

1293 7.5.1. Assertions

1294 SAML assertions can be used as identity tokens that reference the subject of the assertion. Such assertions are generally
1295 signed for integrity, and often contain no statements, only a `<saml2:Subject>` element, possibly with conditions that
1296 limit its use.

1297 When privacy is required, a `<saml2:EncryptedID>` element SHOULD be used in the subject. The decrypted
1298 element SHOULD NOT itself be an assertion (as it would be redundant).

1299 If it is unnecessary to reveal the content of the enclosing assertion to the requester, then a
1300 `<saml2:EncryptedAssertion>` element SHOULD be used, as it is simpler for the requester to handle.
1301 Alternatively, a `<saml2:EncryptedID>` element could be returned directly (see the following section).

1302 SAML assertions used as identity tokens MAY contain ID-WSF EPR attributes and credentials, such as for the
1303 associated Principal's Discovery Service or other ID-WSF-based services.

1304 7.5.2. Identifiers

1305 SAML identifier elements (<saml2:BaseID>, <saml2:NameID>, or <saml2:EncryptedID>) can also be used
1306 as identity tokens. Encrypted identifiers, in particular, can contain actual signed assertions, making them potential
1307 carriers of the assertion forms described in the previous section.

1308 However, in cases where privacy is not a consideration and limits on the use of the identifier are not relevant, a
1309 plaintext form may also be useful, particularly as an input token to a mapping request. For example, an SP that shares
1310 an identifier for a principal with an IdP offering an Identity Mapping service might use a <saml2:NameID> by itself
1311 as an input token.

1312 7.6. Security and Privacy Considerations

1313 Privacy is a critical consideration in the operation of the Identity Mapping Service, because its primary purpose is
1314 to enable entities to invoke services on behalf of principals without requiring the use of globally unique identifiers.
1315 Because identifiers in ID-WSF are typically scoped to particular providers, care must be exercised when allowing
1316 providers to map between them.

1317 In particular, it is usually the case that the identifiers returned by the IMS SHOULD be encrypted when returned
1318 to parties other than those with prior knowledge of them. The use of XML encryption generally results in unique
1319 ciphertext each time a particular value is encrypted, preventing correlation by parties without access to the underlying
1320 plaintext.

1321 It is also important for the IMS to take into consideration the relationship between the input and output tokens. Under
1322 most circumstances, it should not be possible for a requester to map to its own namespace from another, as this would
1323 permit the requester to correlate the original identifier to one that it knows. Rather, the IMS is more generally used to
1324 map from a known identifier into another entity's namespace, returning the result in an encrypted form so that it can
1325 be decrypted by that entity.

1326 7.7. Example Identity Mapping Exchange

1327 The following example shows a request for a SAML identity token. The policy and input token indicate a request to
1328 map from an identifier scoped to one SP into an identifier scoped to another. In this case, the input token is a bare
1329 identifier (probably extracted from another SAML token).

```
1330  
1331 <sa:IdentityMappingRequest>  
1332   <sa:MappingInput>  
1333     <sec:TokenPolicy>  
1334       <samlp2:NameIDPolicy Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"  
1335         SPNameQualifier="https://spb.example.com"/>  
1336     </sec:TokenPolicy>  
1337     <sec:Token>  
1338       <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"  
1339         NameQualifier="https://idp.example.com" SPNameQualifier="https://spa.example.com">  
1340         DBC63923-C718-4249-83CE-1E53D80D8A4A  
1341       </saml2:NameID>  
1342     </sec:Token>  
1343   </sa:MappingInput>  
1344 </sa:IdentityMappingRequest>
```

1345 The following is a possible response to the request above. The returned token is a signed SAML assertion embedded
1346 within an encrypted SAML identifier. The requester can establish the expiration from the response, giving it guidance
1347 as to when the token might need renewal.

```
1348
1349 <sa:IdentityMappingResponse>
1350   <sa:Status code="OK"/>
1351   <sa:MappingOutput>
1352     <sec:Token>
1353       <saml2:EncryptedID>
1354         <xenc:EncryptedData>U2XTCNvRX7B11NK182nmY00TEk==</xenc:EncryptedData>
1355         <xenc:EncryptedKey Recipient="https://spb.example.com">...</xenc:EncryptedKey>
1356       </saml2:EncryptedID>
1357     </sec:Token>
1358   </sa:MappingOutput>
1359 </sa:IdentityMappingResponse>
```

1360

Example 11.

1361 8. Password Transformations: The PasswordTransforms Element

1362 This section defines the <PasswordTransforms> element. Authentication servers MAY use this element to convey
 1363 password pre-processing obligations to clients.

1364 For example, an authentication server may have been configured such that it presumes that the strings users enter as
 1365 their passwords have been pre-processed in some fashion before being further processed and/or stored. For example
 1366 the passwords may be truncated to a given length, and all upper case characters may be folded to lower case, and
 1367 whitespace may be eliminated. The authentication server can communicate these requirements dynamically to clients
 1368 using the <PasswordTransforms> element in an initial <SASLResponse>. See [Figure 8](#).

```

1369
1370 <xs:element name="PasswordTransforms">
1371
1372   <xs:annotation>
1373     <xs:documentation>
1374       Contains ordered list of sequential password transformations
1375     </xs:documentation>
1376   </xs:annotation>
1377
1378   <xs:complexType>
1379     <xs:sequence>
1380
1381       <xs:element name="Transform" maxOccurs="unbounded">
1382         <xs:complexType>
1383           <xs:sequence>
1384
1385             <xs:element name="Parameter"
1386               minOccurs="0"
1387               maxOccurs="unbounded">
1388               <xs:complexType>
1389                 <xs:simpleContent>
1390                   <xs:extension base="xs:string">
1391                     <xs:attribute name="name"
1392                       type="xs:string"
1393                       use="required"/>
1394                   </xs:extension>
1395                 </xs:simpleContent>
1396               </xs:complexType>
1397             </xs:element>
1398
1399           </xs:sequence>
1400
1401           <xs:attribute name="name"
1402             type="xs:anyURI"
1403             use="required"/>
1404
1405           <xs:anyAttribute namespace="##other" processContents="lax"/>
1406
1407         </xs:complexType>
1408       </xs:element>
1409     </xs:sequence>
1410   </xs:complexType>
1411 </xs:element>
1412

```

1413 **Figure 8. The PasswordTransforms element**

```
1414
1415 <PasswordTransforms>
1416   <Transform name="urn:liberty:sa:pm:truncate">
1417     <Parameter name="length">8</Parameter>
1418   </Transform>
1419   <Transform name="urn:liberty:sa:pm:lowercase" />
1420 </PasswordTransforms>
1421
```

1422 **Figure 9. Example of a PasswordTransforms**

1423 Servers **MAY** include a <PasswordTransforms> element along with their **initial** <SASLResponse> to a client. A
1424 <PasswordTransforms> element contains one or more <Transform> elements. Each <Transform> is identified
1425 by the value of the name attribute which must be a URI [RFC3986]. This URI **MUST** specify a particular transforma-
1426 tion on the password. Transforms are specified elsewhere, for example in configuration data at implementation- and/or
1427 deployment-time. A basic set is specified in [Appendix B: Password Transformations](#).

1428 A client receiving an **initial** <SASLResponse> message containing a <PasswordTransforms> element **MUST**
1429 apply the specified transformations to any password that is used as input for the SASL mechanism indicated in the
1430 <SASLResponse>.

1431 The client **MUST** apply the transformations in the order given in the <PasswordTransforms> element, and **MUST**
1432 apply each transform to the result of the preceding transform. Of course, the first transform **MUST** be applied to the
1433 raw password.

1434 Unless the specification of a <Transform> states otherwise, it is specified in terms of [Unicode] *abstract characters*.
1435 An abstract character is a character as rendered to a user. Since an abstract character may require more than one
1436 octet to represent, there is not necessarily a one-to-one mapping between an abstract character, or sequence of abstract
1437 characters, and its corresponding *coded character representation*.

1438 For example, if a truncation transform indicates, "truncate after the first eight characters", the characters after the
1439 eighth abstract character should be removed; in some languages and character encodings this could mean that more
1440 than 8 octets remain.

1441 See also [Appendix B](#).

1442 **9. Acknowledgments**

1443 This spec leverages techniques and ideas from draft-nystrom-http-sasl-xx (an IETF Internet-Draft), RFC3080,
1444 RFC2251, RFC2829, RFC2830, et al (all are various IETF Requests For Comments). The authors of those specs
1445 are gratefully acknowledged. Thanks also to Alexy Melnikov, Paul Madsen, and RL "Bob" Morgan for their feedback
1446 and insights. The docbook source code for this specification was hand set to the tunes of Brad, Bob Mould, Weather
1447 Report, Miles Davis, John Coltrane, Liz Phair, The Wallflowers, Alan Holdsworth, Chick Corea, Jennifer Trynin, Elisa
1448 Korenne, The Cowboy Junkies, Fugazi, Blues Traveler, Blink-182, CSN, Pearl Jam, and various others. Thanks also
1449 to whatever deities are responsible for the existence of coffee, dark chocolate, and fermented cereals.

1450 References

1451 Normative

- 1452 [LibertyAuthnContext] Madsen, Paul, eds. "Liberty ID-FF Authentication Context Specification," Version 2.0-01,
1453 Liberty Alliance Project (21 November 2004). <http://www.projectliberty.org/specs>
- 1454 [LibertyClientProfiles] Aarts, Robert, Kainulainen, Jukka, Kemp, John, eds. Version v2.0, Liberty Alliance Project
1455 (30 July, 2006). <http://www.projectliberty.org/specs>
- 1456 [LibertyDisco] Hodges, Jeff, Cahill, Conor, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0,
1457 Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1458 [LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0,
1459 Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1460 [LibertyPAOS] Aarts, Robert, Kemp, John, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version
1461 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1462 [LibertyPeopleService] Koga, Yuzo, Madsen, Paul, eds. "Liberty ID-WSF People Service Specification," Version 1.0,
1463 Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1464 [LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version
1465 1.2-errata-v3.0, Liberty Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 1466 [LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version v2.0, Liberty
1467 Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1468 [LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0, Liberty Alliance Project (30 July,
1469 2006). <http://www.projectliberty.org/specs>
- 1470 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Lib-
1471 erty ID-WSF SOAP Binding Specification," Version 2.0, Liberty Alliance Project (30 July, 2006).
1472 <http://www.projectliberty.org/specs>
- 1473 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
1474 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>
- 1475 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., eds. (June
1476 1999). "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, The Internet Engineering Task Force
1477 <http://www.ietf.org/rfc/rfc2616.txt>
- 1478 [RFC3066] Alvestrand, H., eds. (January 2001). "Tags for the Identification of Languages," RFC 3066., Internet
1479 Engineering Task Force <http://www.ietf.org/rfc/rfc3066.txt>
- 1480 [RFC3986] Berners-Lee, T., Fielding, R., Masinter, L., eds. (January 2005). "Uniform Resource Identifier
1481 (URI): Generic Syntax," RFC 3986 (Obsoletes RFC2732, RFC2396, RFC1808) (Updates RFC1738) (Also
1482 STD0066) (Status: STANDARD), The Internet Engineering Task Force <http://www.ietf.org/rfc/rfc3986.txt>
- 1483 [RFC4346] Dierks, T., Rescorla, E., eds. (April 2006). "The Transport Layer Security (TLS) Protocol," Version 1.1
1484 RFC 4346, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc4346.txt>
- 1485 [RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., Wright, T., eds. (April 2006).
1486 "Transport Layer Security (TLS) Extensions," RFC 4366, The Internet Engineering Task Force
1487 <http://www.ietf.org/rfc/rfc4366.txt>

- 1488 [RFC4422] "Simple Authentication and Security Layer (SASL)," Melnikov, A., Zeilenga, K., eds. (June 2006). RFC
1489 4422, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc4422.txt>
- 1490 [SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (2 September 2003). "Assertions and Pro-
1491 tocol for the OASIS Security Assertion Markup Language (SAML) V1.1," SAML v1.1, OASIS
1492 Standard, Organization for the Advancement of Structured Information Standards [http://www.oasis-
1493 open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf](http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf)
- 1494 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions
1495 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
1496 SIS Standard, Organization for the Advancement of Structured Information Standards [http://docs.oasis-
1497 open.org/security/saml/v2.0/saml-core-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf)
- 1498 [SAMLGloss2] Hodges, Jeff, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Glossary for the OASIS Security As-
1499 ssertion Markup Language (SAML) V2.0," SAML 2.0, OASIS Standard, Organization for the Advancement
1500 of Structured Information Standards <http://docs.oasis-open.org/security/saml/v2.0/saml-glossary-2.0-os.pdf>
- 1501 [SAMLMeta2] Cantor, Scott, Moreh, Jahan, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Metadata for the OA-
1502 SIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for
1503 the Advancement of Structured Information Standards [http://docs.oasis-open.org/security/saml/v2.0/saml-
1504 metadata-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf)
- 1505 [SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler,
1506 Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0,"
1507 SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards
1508 <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>
- 1509 [SASLReg] "Simple Authentication and Security Layer (SASL) Mechanisms," Internet Assigned Numbers Authority
1510 (IANA) <http://www.iana.org/assignments/sasl-mechanisms>
- 1511 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October
1512 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium
1513 <http://www.w3.org/TR/xmlschema-1/>
- 1514 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David, Kakivaya, Gopal, Layman,
1515 Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C
1516 Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 1517 [Unicode] The Unicode Consortium (2003). "The Unicode Standard, version 4.0," Addison-Wesley *Unicode 4.0.0*
1518 [<http://www.unicode.org>]
- 1519 [WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, Rogers, Tony, eds.
1520 World Wide Web Consortium W3C Recommendation (9 May 2006). [http://www.w3.org/TR/2006/REC-ws-
1521 addr-core-20060509/](http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/)
- 1522 [WSAv1.0-SOAP] "WS-Addressing 1.0 SOAP Binding," Gudgin, Martin, Hadley, Marc, eds. World Wide Web Con-
1523 sultium W3C Recommendation (9 May 2006). <http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/>
- 1524 [WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Meredith,
1525 Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001).
1526 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

1527 Informational

- 1528 [IANA] "The Internet Assigned Numbers Authority," <http://www.iana.org/>
- 1529 [LibertyIDPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification,"
1530 Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 1531 [LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework
1532 Overview," Version 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1533 [Merriam-Webster] "Merriam-Webster Dictionary," <http://www.merriam-webster.com/>
- 1534 [RFC2289] "A One-Time Password System," N. Haller C. Metz P. Nessner M. Straw (February 1998). RFC 2289,
1535 Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2289.txt>
- 1536 [RFC2444] Newman, C., eds. (October 1998). "The One-Time-Password SASL Mechanism," RFC 2444, The Internet
1537 Engineering Task Force <http://www.ietf.org/rfc/rfc2444.txt>
- 1538 [RFC2828] Shirey, R., eds. (May 2000). "Internet Security Glossary," RFC 2828., Internet Engineering Task Force
1539 <http://www.ietf.org/rfc/rfc2828.txt>
- 1540 [RFC3163] Zuccherato, R., Nystrom, M., eds. (August 2001). "ISO/IEC 9798-3 Authentication SASL Mechanism,"
1541 RFC 3163, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc3163.txt>
- 1542 [SAMLBind2] Cantor, Scott, Hirsch, Frederick, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March
1543 2005). "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
1544 SIS Standard, Organization for the Advancement of Structured Information Standards [http://docs.oasis-
1545 open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf)
- 1546 [SOAPv1.2] "SOAP Version 1.2 Part 1: Messaging Framework," Gudgin, Martin, Hadley, Marc, Mendelsohn, Noah,
1547 Moreau, Jean-Jacques, Nielsen, Henrik Frystyk, eds. World Wide Web Consortium W3C Recommendation
1548 (07 May 2003). <http://www.w3.org/TR/2003/PR-soap12-part1-20030507/>
- 1549 [TrustInCyberspace] Schneider, Fred B., eds. "Trust in Cyberspace," National Research Council (1999).
1550 <http://www.nap.edu/readingroom/books/trust/>
- 1551 [WooLam92] Thomas Y. C. Woo Simon S. Lam "Authentication for Distributed Systems," (Jan-
1552 uary, 1992). IEEE Computer Society IEEE Computer (Vol. 25, No. 1), pp. 39-52
1553 <http://doi.ieeecomputersociety.org/10.1109/2.108052>
- 1554 [wss-sms] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (January, 2004). "Web
1555 Services Security: SOAP Message Security," OASIS Standard V1.0 [OASIS 200401], Organization for
1556 the Advancement of Structured Information Standards [http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-
1557 wss-soap-message-security-1.0.pdf)
- 1558 [wss-saml11] Monzillo, Ronald, Kaler, Chris, Nadalin, Anthony, Hallam-Baker, Phillip, eds. (June 28,
1559 2005). Organization for the Advancement of Structured Information Standards [http://www.oasis-
1561 open.org/committees/download.php/13405/wss-v1.1-spec-pr-SAMLTokenProfile-01.pdf](http://www.oasis-
1560 open.org/committees/download.php/13405/wss-v1.1-spec-pr-SAMLTokenProfile-01.pdf) "Web Services
Security: SAML Token Profile 1.1," OASIS Public Review Draft 01,
- 1562 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).
1563 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium
1564 <http://www.w3.org/TR/2004/REC-xml-20040204>

1565 **A. Listing of Simple Authentication and Security Layer (SASL)**
1566 **Mechanisms**

1567 Ref: [[SASLReg](#)]

1568 **Note**

1569 The file listed below IS SUBJECT TO CHANGE! It is presented here as non-normative background information only.
1570 Implementers and deployers should always retrieve a fresh copy of this file from [[IANA](#)].

1571

1572 SIMPLE AUTHENTICATION AND SECURITY LAYER (SASL) MECHANISMS

1573 -----

1574

1575 (last updated 15 May 2006)

1576

1577 The Simple Authentication and Security Layer (SASL) [RFC-ietf-sasl-rfc2222bis-15.txt] is a
1578 method for adding authentication support to connection-based
1579 protocols. To use this specification, a protocol includes a command
1580 for identifying and authenticating a user to a server and for
1581 optionally negotiating a security layer for subsequent protocol
1582 interactions. The command has a required argument identifying a SASL
1583 mechanism.

1584

1585 SASL mechanisms are named by strings, from 1 to 20 characters in
1586 length, consisting of upper-case letters, digits, hyphens, and/or
1587 underscores. SASL mechanism names must be registered with the IANA.
1588 Procedures for registering new SASL mechanisms are described in
1589 RFC-ietf-sasl-rfc2222bis-15.txt.

1590

1591 Registration Procedures:

1592 First Come First Serve for Mechanisms

1593 Expert Review with Mailing List for Family Name Registrations

1594

1595 MECHANISMS USAGE REFERENCE OWNER

1596 ----- - - -

1597 KERBEROS_V4 OBSOLETE [RFC2222] IESG <iesg@ietf.org>

1598

1599 GSSAPI COMMON [RFC2222] IESG <iesg@ietf.org>

1600

1601 SKEY OBSOLETE [RFC2444] IESG <iesg@ietf.org>

1602

1603 EXTERNAL COMMON [RFC-ietf-sasl-rfc2222bis-15.txt] IESG <iesg@ietf.org>

1604

1605 CRAM-MD5 LIMITED [RFC2195] IESG <iesg@ietf.org>

1606

1607 ANONYMOUS COMMON [RFC-ietf-sasl-anon-05.txt] IESG <iesg@ietf.org>

1608

1609 OTP COMMON [RFC2444] IESG <iesg@ietf.org>

1610

1611 GSS-SPNEGO LIMITED [Leach] Paul Leach <paulle@microsoft.com>

1612

1613 PLAIN COMMON [RFC2595] IESG <iesg@ietf.org>

1614

1615 SECURID COMMON [RFC2808] Magnus Nystrom <magnus@rsasecurity.com>

1616

1617 NTLM LIMITED [Leach] Paul Leach <paulle@microsoft.com>

1618

1619 NMAS_LOGIN LIMITED [Gayman] Mark G. Gayman <mgayman@novell.com>

1620

1621 NMAS_AUTHEN LIMITED [Gayman] Mark G. Gayman <mgayman@novell.com>

1622

1623 DIGEST-MD5 COMMON [RFC2831] IESG <iesg@ietf.org>

1624

1625 9798-U-RSA-SHA1-ENC COMMON [RFC3163] robert.zuccherato@entrust.com

1626
1627 9798-M-RSA-SHA1-ENC COMMON [RFC3163] robert.zuccherato@entrust.com
1628
1629 9798-U-DSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com
1630
1631 9798-M-DSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com
1632
1633 9798-U-ECDSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com
1634
1635 9798-M-ECDSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com
1636
1637 KERBEROS_V5 COMMON [Josefsson] Simon Josefsson <simon@josefsson.org>
1638
1639 NMAS-SAMBA-AUTH LIMITED [Brimhall] Vince Brimhall <vbrimhall@novell.com>
1640
1641
1642 References
1643 -----
1644 [RFC2195] Klensin, J., Catoe, R., Krumviede, P. "IMAP/POP AUTHorize
1645 Extension for Simple Challenge/Response", RFC 2195, MCI,
1646 September 1997.
1647
1648 [RFC2222] J. Myers, "Simple Authentication and Security Layer (SASL)",
1649 RFC 2222, October 1997.
1650
1651 [RFC2444] Newman, C., "The One-Time-Password SASL Mechanism", RFC
1652 2444, October 1998.
1653
1654 [RFC2595] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595,
1655 Innosoft, June 1999.
1656
1657 [RFC2808] Nystrom, M., "The SecurID(r) SASL Mechanism", RFC 2808,
1658 April 2000.
1659
1660 [RFC2831] Leach, P. and C. Newman, "Using Digest Authentication as a
1661 SASL Mechanism", RFC 2831, May 2000.
1662
1663 [RFC3163] R. Zuccherato and M. Nystrom, "ISO/IEC 9798-3 Authentication
1664 SASL Mechanism", RFC 3163, August 2001.
1665
1666 [RFC-ietf-sasl-anon-05.txt]
1667 K. Zeilenga, Ed., "The Anonymous SASL Mechanism", RFC XXXX,
1668 Month Year.
1669
1670 [RFC-ietf-sasl-rfc2222bis-15.txt]
1671 A. Melnikov and K. Zeilenga, "Simple Authentication and Security
1672 Layer (SASL)", RFC XXXX, Month Year.
1673
1674 People
1675 -----
1676
1677 [Brimhall] Vince Brimhall, <vbrimhall@novell.com>, April 2004.
1678
1679 [Gayman] Mark G. Gayman, <mgayman@novell.com>, September 2000.
1680
1681 [Josefsson] Simon Josefsson, <simon@josefsson.org>, January 2004.
1682
1683 [Leach] Paul Leach, <paulle@microsoft.com>, December 1998, June 2000.
1684
1685 []

1686 **B. Password Transformations**

1687 This section defines a number of password transformations.

1688 **1. Truncation**

1689 The `urn:liberty:sa:pw:truncate` transformation instructs processors to remove all (Unicode abstract) sub-
1690 sequent characters after a given number of characters have been obtained (from the user). Subsequent processing
1691 MUST take only the given number of characters as input. The number of characters that shall remain is given in a
1692 `<Parameter>` element with name "length".

```
1693  
1694 <Transform name="urn:liberty:sa:pw:truncate">  
1695   <Parameter name="length">8</Parameter>  
1696 </Transform>  
1697
```

1698 **Figure B.1. Example of truncation transformation**

1699 **2. Lowercase**

1700 The `urn:liberty:sa:pw:lowercase` transformation instructs processors to replace all uppercase characters with
1701 lowercase characters. Characters that do not have case must remain unchanged. This transformation has no parameters.
1702 Note that the "case" of the abstract Unicode character is decisive, i.e. only characters that have the `UPPERCASE`
1703 property should be replaced with equivalent characters with the `Lowercase` property. This mapping from `UPPERCASE`
1704 to `lowercase` should confirm to the relevant sections (e.g. 4.2) of [Unicode].

```
1705  
1706 <Transform name="urn:liberty:sa:pw:lowercase" />  
1707
```

1708 **Figure B.2. Example of lowercase transformation**

1709 **3. Uppercase**

1710 The `urn:liberty:sa:pw:uppercase` transformation instructs processors to replace all lowercase characters with
1711 uppercase characters. Characters that do not have case must remain unchanged. This transformation has no parameters.
1712 Note that the "case" of the abstract Unicode character is decisive, i.e. only characters that have the `Lowercase`
1713 property should be replaced with equivalent characters with the `Uppercase` property. This mapping from `lowercase`
1714 to `UPPERCASE` should confirm to the relevant sections (e.g. 4.2) of [Unicode].

```
1715  
1716 <Transform name="urn:liberty:sa:pw:uppercase" />  
1717
```

1718 **Figure B.3. Example of uppercase transformation**

1719 **4. Select**

1720 The `urn:liberty:sa:pw:select` transformation instructs processors to remove all characters except those spec-
1721 ified in the "allowed" parameter. Note that the allowed characters refer to abstract Unicode characters. In the
1722 message that contains the `<Transform>` element these characters are encoded with the same encoding as used for the
1723 xml document that contains the message (usually UTF-8).

```
1724  
1725 <Transform name="urn:liberty:sa:pw:select">  
1726   <Parameter name="allowed">0123456789abcdefghijklmnopqrstvwxyz</Parameter>  
1727 </Transform>  
1728
```

1729 **Figure B.4. Example of select transformation**

1730 C. liberty-idwsf-authn-svc-v2.0.xsd Schema Listing

```
1731 <?xml version="1.0" encoding="UTF-8"?>
1732
1733 <xs:schema
1734     targetNamespace="urn:liberty:sa:2006-08"
1735     xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1736     xmlns:sa="urn:liberty:sa:2006-08"
1737     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1738     xmlns:samlp2="urn:oasis:names:tc:SAML:2.0:protocol"
1739     xmlns:wsa="http://www.w3.org/2005/08/addressing"
1740     xmlns:lu="urn:liberty:util:2006-08"
1741     xmlns="urn:liberty:sa:2006-08"
1742     elementFormDefault="qualified"
1743     attributeFormDefault="unqualified"
1744     version="09"
1745 >
1746
1747 <!-- Filename: lib-arch-authn-svc.xsd -->
1748 <!-- $Id: lib-arch-authn-svc.xsd,v 1.22.2.1 2006/07/28 02:44:19 dchampagne Exp $ -->
1749 <!-- Author: Jeff Hodges -->
1750 <!-- Last editor: $Author: dchampagne $ -->
1751 <!-- $Date: 2006/07/28 02:44:19 $ -->
1752 <!-- $Revision: 1.22.2.1 $ -->
1753
1754 <xs:import
1755     namespace="http://www.w3.org/2005/08/addressing"
1756     schemaLocation="ws-addr-1.0.xsd"/>
1757
1758 <xs:import
1759     namespace="urn:oasis:names:tc:SAML:2.0:protocol"
1760     schemaLocation="saml-schema-protocol-2.0.xsd"/>
1761
1762 <xs:import namespace="urn:liberty:util:2006-08"
1763     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
1764
1765 <xs:annotation>
1766     <xs:documentation>
1767         Liberty ID-WSF Authentication Service XSD
1768     </xs:documentation>
1769     <xs:documentation>
1770         The source code in this XSD file was excerpted verbatim from:
1771
1772         Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification
1773         Version 2.0
1774         30 July, 2006
1775
1776         Copyright (c) 2006 Liberty Alliance participants,
1777         see http://www.projectliberty.org/specs/idwsf\_2\_0\_copyrights.php
1778     </xs:documentation>
1779 </xs:annotation>
1780
1781
1782 <!-- SASLRequest and SASLResponse ID-* messages -->
1783
1784 <xs:element name="SASLRequest">
1785     <xs:complexType>
1786         <xs:sequence>
1787
1788             <xs:element name="Data" minOccurs="0">
1789                 <xs:complexType>
1790                     <xs:simpleContent>
1791                         <xs:extension base="xs:base64Binary"/>
1792                     </xs:simpleContent>
1793                 </xs:complexType>
1794             </xs:element>
1795
```

```
1796         <xs:element ref="samlp2:RequestedAuthnContext" minOccurs="0" />
1797
1798         <xs:element name="Extensions" minOccurs="0">
1799             <xs:complexType>
1800                 <xs:sequence>
1801                     <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded" />
1802                 </xs:sequence>
1803             </xs:complexType>
1804         </xs:element>
1805
1806     </xs:sequence>
1807
1808     <xs:attribute name="mechanism"
1809                 type="xs:string"
1810                 use="required" />
1811
1812     <xs:attribute name="authzID"
1813                 type="xs:string"
1814                 use="optional" />
1815
1816     <xs:attribute name="advisoryAuthnID"
1817                 type="xs:string"
1818                 use="optional" />
1819
1820     <xs:anyAttribute namespace="##other" processContents="lax" />
1821
1822 </xs:complexType>
1823 </xs:element>
1824
1825 <xs:element name="SASLResponse">
1826     <xs:complexType>
1827         <xs:sequence>
1828
1829             <xs:element ref="lu:Status" />
1830
1831             <xs:element ref="PasswordTransforms" minOccurs="0" />
1832
1833             <xs:element name="Data" minOccurs="0">
1834                 <xs:complexType>
1835                     <xs:simpleContent>
1836                         <xs:extension base="xs:base64Binary" />
1837                     </xs:simpleContent>
1838                 </xs:complexType>
1839             </xs:element>
1840
1841             <!-- ID-WSF EPRs -->
1842             <xs:element ref="wsa:EndpointReference"
1843                     minOccurs="0"
1844                     maxOccurs="unbounded" />
1845
1846         </xs:sequence>
1847
1848         <xs:attribute name="serverMechanism"
1849                     type="xs:string"
1850                     use="optional" />
1851
1852         <xs:anyAttribute namespace="##other" processContents="lax" />
1853     </xs:complexType>
1854 </xs:element>
1855
1856
1857
1858 <!-- Password Transformations -->
1859
1860 <xs:element name="PasswordTransforms">
1861
1862     <xs:annotation>
```

```
1863     <xs:documentation>
1864         Contains ordered list of sequential password transformations
1865     </xs:documentation>
1866 </xs:annotation>
1867
1868 <xs:complexType>
1869     <xs:sequence>
1870
1871         <xs:element name="Transform" maxOccurs="unbounded">
1872             <xs:complexType>
1873                 <xs:sequence>
1874
1875                     <xs:element name="Parameter"
1876                         minOccurs="0"
1877                         maxOccurs="unbounded">
1878                         <xs:complexType>
1879                             <xs:simpleContent>
1880                                 <xs:extension base="xs:string">
1881                                     <xs:attribute name="name"
1882                                         type="xs:string"
1883                                         use="required" />
1884                                 </xs:extension>
1885                             </xs:simpleContent>
1886                         </xs:complexType>
1887                     </xs:element>
1888
1889                 </xs:sequence>
1890
1891                 <xs:attribute name="name"
1892                     type="xs:anyURI"
1893                     use="required" />
1894
1895                 <xs:anyAttribute namespace="##other" processContents="lax" />
1896
1897             </xs:complexType>
1898         </xs:element>
1899     </xs:sequence>
1900 </xs:complexType>
1901 </xs:element>
1902
1903 </xs:schema>
1904
1905
```

1906 D. liberty-idwsf-idmapping-svc-v2.0.xsd Schema Listing

```
1907 <?xml version="1.0" encoding="UTF-8"?>
1908
1909 <xs:schema
1910     targetNamespace="urn:liberty:ims:2006-08"
1911     xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1912     xmlns:ims="urn:liberty:ims:2006-08"
1913     xmlns:sec="urn:liberty:security:2006-08"
1914     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1915     xmlns:lu="urn:liberty:util:2006-08"
1916     xmlns="urn:liberty:ims:2006-08"
1917     elementFormDefault="qualified"
1918     attributeFormDefault="unqualified"
1919 >
1920
1921 <!-- Filename: liberty-idwsf-idmapping-svc-v2.0.xsd -->
1922 <!-- $Id: lib-arch-idmapping-svc.xsd,v 1.7.2.2 2006/07/28 02:44:19 dchampagne Exp $ -->
1923 <!-- Author: Scott Cantor -->
1924 <!-- Last editor: $Author: dchampagne $ -->
1925 <!-- $Date: 2006/07/28 02:44:19 $ -->
1926 <!-- $Revision: 1.7.2.2 $ -->
1927
1928 <xs:import
1929     namespace="urn:liberty:security:2006-08"
1930     schemaLocation="liberty-idwsf-security-mechanisms-v2.0.xsd"/>
1931
1932 <xs:import namespace="urn:liberty:util:2006-08"
1933     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
1934
1935 <xs:annotation>
1936     <xs:documentation>
1937         Liberty ID-WSF Identity Mapping Service XSD
1938     </xs:documentation>
1939     <xs:documentation>
1940         The source code in this XSD file was excerpted verbatim from:
1941
1942         Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification
1943         Version 2.0
1944         30 July, 2006
1945
1946         Copyright (c) 2006 Liberty Alliance participants,
1947         see http://www.projectliberty.org/specs/idwsf\_2\_0\_copyrights.php
1948     </xs:documentation>
1949 </xs:annotation>
1950
1951 <xs:element name="MappingInput" type="MappingInputType"/>
1952 <xs:complexType name="MappingInputType">
1953     <xs:sequence>
1954         <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
1955         <xs:element ref="sec:Token" minOccurs="0"/>
1956     </xs:sequence>
1957     <xs:attribute name="reqID" type="lu:IDType" use="optional"/>
1958 </xs:complexType>
1959
1960 <xs:element name="MappingOutput" type="MappingOutputType"/>
1961 <xs:complexType name="MappingOutputType">
1962     <xs:sequence>
1963         <xs:element ref="sec:Token"/>
1964     </xs:sequence>
1965     <xs:attribute name="reqRef" type="lu:IDReferenceType" use="optional"/>
1966 </xs:complexType>
1967
1968 <xs:element name="IdentityMappingRequest" type="IdentityMappingRequestType"/>
1969 <xs:complexType name="IdentityMappingRequestType">
1970     <xs:sequence>
1971         <xs:element ref="MappingInput" maxOccurs="unbounded"/>
```

```
1972     </xs:sequence>
1973     <xs:anyAttribute namespace="##other" processContents="lax" />
1974 </xs:complexType>
1975
1976     <xs:element name="IdentityMappingResponse" type="IdentityMappingResponseType" />
1977 <xs:complexType name="IdentityMappingResponseType">
1978     <xs:sequence>
1979         <xs:element ref="lu:Status" />
1980         <xs:element ref="MappingOutput" minOccurs="0" maxOccurs="unbounded" />
1981     </xs:sequence>
1982     <xs:anyAttribute namespace="##other" processContents="lax" />
1983 </xs:complexType>
1984
1985 </xs:schema>
1986
1987
```

1988 E. liberty-idwsf-utility-v2.0.xsd Schema Listing

```
1989 <?xml version="1.0" encoding="UTF-8"?>
1990 <xs:schema targetNamespace="urn:liberty:util:2006-08"
1991   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1992   xmlns="urn:liberty:util:2006-08"
1993   elementFormDefault="qualified"
1994   attributeFormDefault="unqualified"
1995   version="2.0-03">
1996
1997   <xs:annotation>
1998     <xs:documentation>
1999       Liberty Alliance Project utility schema. A collection of common
2000       IDentity Web Services Framework (ID-WSF) elements and types.
2001       This schema is intended for use in ID-WSF schemas.
2002
2003       This version: 2006-08
2004
2005       Copyright (c) 2006 Liberty Alliance participants, see
2006       http://www.projectliberty.org/specs/idwsf\_2\_0\_final\_copyrights.php
2007
2008     </xs:documentation>
2009   </xs:annotation>
2010   <xs:simpleType name="IDType">
2011     <xs:annotation>
2012       <xs:documentation>
2013         This type should be used to provide IDs to components
2014         that have IDs that may not be scoped within the local
2015         xml instance document.
2016       </xs:documentation>
2017     </xs:annotation>
2018     <xs:restriction base="xs:string"/>
2019   </xs:simpleType>
2020   <xs:simpleType name="IDReferenceType">
2021     <xs:annotation>
2022       <xs:documentation>
2023         This type can be used when referring to elements that are
2024         identified using an IDType.
2025       </xs:documentation>
2026     </xs:annotation>
2027     <xs:restriction base="xs:string"/>
2028   </xs:simpleType>
2029   <xs:attribute name="itemID" type="IDType"/>
2030   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2031   <xs:complexType name="StatusType">
2032     <xs:annotation>
2033       <xs:documentation>
2034         A type that may be used for status codes.
2035       </xs:documentation>
2036     </xs:annotation>
2037     <xs:sequence>
2038       <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
2039     </xs:sequence>
2040     <xs:attribute name="code" type="xs:string" use="required"/>
2041     <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
2042     <xs:attribute name="comment" type="xs:string" use="optional"/>
2043   </xs:complexType>
2044
2045   <xs:element name="Status" type="StatusType">
2046     <xs:annotation>
2047       <xs:documentation>
2048         A standard Status type
2049       </xs:documentation>
2050     </xs:annotation>
2051   </xs:element>
2052
2053   <xs:complexType name="ResponseType">
```

```
2054     <xs:sequence>
2055         <xs:element ref="Status" minOccurs="1" maxOccurs="1"/>
2056         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2057     </xs:sequence>
2058     <xs:attribute ref="itemIDRef" use="optional"/>
2059     <xs:anyAttribute namespace="##other" processContents="lax"/>
2060 </xs:complexType>
2061 <xs:element name="TestResult" type="TestResultType"/>
2062 <xs:complexType name="TestResultType">
2063     <xs:simpleContent>
2064         <xs:extension base="xs:boolean">
2065             <xs:attribute ref="itemIDRef" use="required"/>
2066         </xs:extension>
2067     </xs:simpleContent>
2068 </xs:complexType>
2069 <xs:complexType name="EmptyType">
2070     <xs:annotation>
2071         <xs:documentation> This type may be used to create an empty element </xs:documentation>
2072     </xs:annotation>
2073     <xs:complexContent>
2074         <xs:restriction base="xs:anyType"/>
2075     </xs:complexContent>
2076 </xs:complexType>
2077 <xs:element name="Extension" type="extensionType">
2078     <xs:annotation>
2079         <xs:documentation>
2080             An element that contains arbitrary content extensions
2081             from other namespaces
2082         </xs:documentation>
2083     </xs:annotation>
2084 </xs:element>
2085 <xs:complexType name="extensionType">
2086     <xs:annotation>
2087         <xs:documentation>
2088             A type for arbitrary content extensions from other namespaces
2089         </xs:documentation>
2090     </xs:annotation>
2091     <xs:sequence>
2092         <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
2093     </xs:sequence>
2094 </xs:complexType>
2095 </xs:schema>
2096
```

2097 F. liberty-idwsf-authn-svc-v2.0.wsdl WSDL Listing

```
2098 <?xml version="1.0"?>
2099 <definitions name="AuthenticationService"
2100     targetNamespace="urn:liberty:sa:2006-08"
2101     xmlns:tns="urn:liberty:sa:2006-08"
2102     xmlns:xs="http://www.w3.org/2001/XMLSchema"
2103     xmlns:S="http://schemas.xmlsoap.org/wsdl/soap/"
2104     xmlns="http://schemas.xmlsoap.org/wsdl/"
2105     xmlns:sa="urn:liberty:sa:2006-08"
2106     xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
2107     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2108     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2109     xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
2110         http://schemas.xmlsoap.org/wsdl/
2111         http://www.w3.org/2006/02/addressing/wsdl
2112         http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
2113
2114     <xsd:documentation>
2115
2116         XML Authentication Schema from Liberty ID-WSF Authentication,
2117         Single Sign-On, and Identity Mapping Services Specification
2118
2119         ### NOTICE ###
2120
2121         Copyright (c) 2006 Liberty Alliance participants, see
2122         http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2123
2124     </xsd:documentation>
2125     <types>
2126         <xs:schema
2127             <xs:import namespace="urn:liberty:sa:2006-08"
2128                 schemaLocation="liberty-idwsf-authn-svc-v2.0.xsd" />
2129         </xs:schema>
2130     </types>
2131
2132     <message name="AuthenticationSoapRequest">
2133         <part name="parameters" element="sa:SASLRequest" />
2134     </message>
2135     <message name="AuthenticationSoapResponse">
2136         <part name="parameters" element="sa:SASLResponse" />
2137     </message>
2138
2139     <portType name="AuthServicePortType">
2140         <operation name="Authenticate">
2141             <input message="sa:AuthenticationSoapRequest"
2142                 wsaw:Action="urn:liberty:sa:2006-08:SASLRequest"/>
2143             <output message="sa:AuthenticationSoapResponse"
2144                 wsaw:Action="urn:liberty:sa:2006-08:SASLResponse"/>
2145         </operation>
2146     </portType>
2147     <binding name="AuthenticationSoapBinding" type="sa:AuthServicePortType">
2148         <S:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2149         <operation name="Authenticate">
2150             <S:operation soapAction="urn:liberty:sa:2006-08:Authenticate" style="document"/>
2151             <input>
2152                 <S:body use="literal" />
2153             </input>
2154             <output>
2155                 <S:body use="literal" />
2156             </output>
2157         </operation>
2158     </binding>
2159     <service name="AuthenticationService">
2160         <port name="AuthServicePortType" binding="sa:AuthenticationSoapBinding">
2161             <S:address location="http://example.com/authentication"/>
2162         </port>
```

```
2163     </service>  
2164 </definitions>  
2165
```

2166 G. liberty-idwsf-ssosvc-v2.0.wsdl WSDL Listing

```
2167 <?xml version="1.0"?>
2168 <definitions name="AuthenticationService"
2169     targetNamespace="urn:liberty:ssos:2006-08 "
2170     xmlns:tns="urn:liberty:ssos:2006-08 "
2171     xmlns:xs="http://www.w3.org/2001/XMLSchema"
2172     xmlns:S="http://schemas.xmlsoap.org/wsdl/soap/"
2173     xmlns="http://schemas.xmlsoap.org/wsdl/"
2174     xmlns:ssos="urn:liberty:ssos:2006-08 "
2175     xmlns:samlp2="urn:oasis:names:tc:SAML:2.0:protocol"
2176     xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
2177     xmlns:xsd="http://www.w3.org/2001/XMLSchema "
2178     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2179     xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
2180         http://schemas.xmlsoap.org/wsdl/
2181         http://www.w3.org/2006/02/addressing/wsdl
2182         http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
2183
2184     <xsd:documentation>
2185
2186         XML SSO Schema from Liberty ID-WSF Authentication, Single
2187         Sign-On, and Identity Mapping Services Specification
2188
2189         ### NOTICE ###
2190
2191         Copyright (c) 2006 Liberty Alliance participants, see
2192         http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2193
2194     </xsd:documentation>
2195     <types>
2196         <xs:schema
2197             <xs:import namespace="urn:oasis:names:tc:SAML:2.0:protocol"
2198                 schemaLocation="saml-schema-protocol-2.0.xsd" />
2199         </xs:schema>
2200     </types>
2201
2202     <message name="SSOSoapRequest">
2203         <part name="parameters" element="samlp2:AuthnRequest" />
2204     </message>
2205     <message name="SSOSoapResponse">
2206         <part name="parameters" element="samlp2:Response" />
2207     </message>
2208
2209     <portType name="SSOSPortType">
2210         <operation name="SingleSignOn">
2211             <input message="ssos:SSOSoapRequest"
2212                 wsaw:Action="urn:liberty:ssos:2006-08:AuthnRequest" />
2213             <output message="ssos:SSOSoapResponse"
2214                 wsaw:Action="urn:liberty:ssos:2006-08:Response" />
2215         </operation>
2216     </portType>
2217     <binding name="SSOSSoapBinding" type="ssos:SSOSPortType">
2218         <S:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2219         <operation name="SingleSignOn">
2220             <S:operation soapAction="urn:liberty:ssos:2006-08:SingleSignOn" style="document"/>
2221             <input>
2222                 <S:body use="literal" />
2223             </input>
2224             <output>
2225                 <S:body use="literal" />
2226             </output>
2227         </operation>
2228     </binding>
2229     <service name="SSOService">
2230         <port name="SSOSPortType" binding="ssos:SSOSSoapBinding">
2231             <S:address location="http://example.com/idmapping"/>

```

```
2232         </port>  
2233     </service>  
2234 </definitions>  
2235
```

2236 H. liberty-idwsf-idmapping-svc-v2.0.wsdl WSDL Listing

```
2237 <?xml version="1.0"?>
2238 <definitions name="AuthenticationService"
2239     targetNamespace="urn:liberty:ims:2006-08"
2240     xmlns:tns="urn:liberty:ims:2006-08"
2241     xmlns:xs="http://www.w3.org/2001/XMLSchema"
2242     xmlns:S="http://schemas.xmlsoap.org/wsdl/soap/"
2243     xmlns="http://schemas.xmlsoap.org/wsdl/"
2244     xmlns:ims="urn:liberty:ims:2006-08"
2245     xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
2246     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2247     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2248     xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
2249         http://schemas.xmlsoap.org/wsdl/
2250         http://www.w3.org/2006/02/addressing/wsdl
2251         http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
2252
2253     <xsd:documentation>
2254
2255         XML ID Mapping Schema from Liberty ID-WSF Authentication, Single
2256         Sign-On, and Identity Mapping Services Specification
2257
2258         ### NOTICE ###
2259
2260         Copyright (c) 2006 Liberty Alliance participants, see
2261         http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2262
2263     </xsd:documentation>
2264     <types>
2265         <xs:schema
2266             <xs:import namespace="urn:liberty:ims:2006-08"
2267                 schemaLocation="liberty-idwsf-idmapping-svc-v2.0.xsd" />
2268         </xs:schema>
2269     </types>
2270
2271     <message name="IdentityMappingSoapRequest">
2272         <part name="parameters" element="ims:IdentityMappingRequest" />
2273     </message>
2274     <message name="IdentityMappingSoapResponse">
2275         <part name="parameters" element="ims:IdentityMappingResponse" />
2276     </message>
2277
2278     <portType name="IdMappingPortType">
2279         <operation name="IdentityMapping">
2280             <input message="ims:IdentityMappingSoapRequest"
2281                 wsaw:Action="urn:liberty:ims:2006-08:IdentityMappingRequest" />
2282             <output message="ims:IdentityMappingSoapResponse"
2283                 wsaw:Action="urn:liberty:ims:2006-08:IdentityMappingResponse" />
2284         </operation>
2285     </portType>
2286     <binding name="IdMappingSoapBinding" type="ims:IdMappingPortType">
2287         <S:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
2288         <operation name="IdentityMapping">
2289             <S:operation soapAction="urn:liberty:ims:2006-08:IdentityMapping" style="document" />
2290             <input>
2291                 <S:body use="literal" />
2292             </input>
2293             <output>
2294                 <S:body use="literal" />
2295             </output>
2296         </operation>
2297     </binding>
2298     <service name="IdMappingService">
2299         <port name="IdMappingPortType" binding="ims:IdMappingSoapBinding">
2300             <S:address location="http://example.com/idmapping" />
2301         </port>
```

```
2302     </service>  
2303 </definitions>  
2304
```