

1



2

3

## 4 **Liberty Basic SOAP Binding**

5 **Version:** 1.0

### 6 **Editors:**

7 Søren Peter Nielsen, Danish National IT and Telecom Agency

8 Thomas Gundel, IT Crew

### 9 **Contributors:**

10 Conor P. Cahill, Intel

11 George Fletcher, AOL

12 Paul Madsen, NTT

13 Sampo Kellomaki, Symlabs

14 Pat Patterson, Sun Microsystems

15 Colin Wallis, New Zealand Government State Services Commission

### 16 **Abstract:**

17 This document contains a basic profile of the Liberty ID-WSF SOAP binding 2.0.

18 **Filename:** Liberty-Basic-SOAP-Binding-1.0.pdf

19 This profile has been developed from business requirements within eGovernment, but is  
20 believed to be generally applicable. Liberty Alliance is making this profile publicly  
21 available to the industry at large for review and consideration.

22 **Notice**

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

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

40 Copyright © 2009 ActivIdentity, Trent Adams, Adetti, Adobe Systems, AOL, BEA  
41 Systems, Berne, University of Applied Sciences, Gerald Beuchelt, BIPAC, John Bradley,  
42 British Telecommunications plc, Hellmuth Broda, Bronnoysund Register Centre, BUPA,  
43 CA, Canada Post Corporation, Center for Democracy and Technology, Chief,  
44 Information Office Austria, China Internet Network Information Center (CNNIC),  
45 ChoicePoint, Citi, City University, Clareity Security, Dan Combs, Computer &  
46 Communications Industry Association, Courion Corporation, Danish Biometrics  
47 Research Proj. Consortium, Danish National IT and Telecom Agency, Deny All,  
48 Deutsche Telekom AG, DGME, Diversinet Corp., Drummond Group Inc., East of  
49 England Telematics Development Trust Ltd, EifEL, Electronics and Telecommunications  
50 Research Institute (ETRI), Engineering Partnership in Lancashire, Enterprise Java  
51 Victoria Inc., Entr'ouvert, Ericsson, eValid8, Evidian, Fidelity Investments, Financial  
52 Services Technology Consortium (FSTC), Finland National Board of Taxes, Fischer  
53 International, France Telecom, Fraunhofer-Gesellschaft, Fraunhofer Institute for  
54 Integrated Circuits IIS, Fraunhofer Institute for Secure Information Technology (SIT),  
55 Fraunhofer Institut for Experimentelles Software Engineering, Fugen Solutions, Fujitsu  
56 Services Oy, Fun Communications GmbH, Gemalto, Giesecke & Devrient GMBH,  
57 Global Platform, GSA Office of Governmentwide Policy, Healthcare Financial

58 Management Association (HFMA), Health Information and Management Systems  
59 Society (HIMSS), Helsinki Institute of Physics, Jeff Hodges, Hongkong Post, Guy  
60 Huntington, Imprivata, Information Card Foundation, Institute of Bioorganic Chemistry  
61 Poland, Institute of Information Management of the University, Institut Experimentelles  
62 Software Engineering (IESE), Intel Corporation, International Institute of  
63 Telecommunications, International Security, Trust and Privacy Alliance, Internet2,  
64 Interoperability Clearinghouse (ICH), ISOC, Java Wireless Competency Centre (JWCC),  
65 Kantega AS, Kuppinger Cole & Partner, Kuratorium OFFIS e.V., Colin Mallett, Rob  
66 Marano, McMaster University, MEDNETWorld.com, Methics Oy, Mortgage Bankers  
67 Association (MBA), Mydex, National Institute for Urban Search & Rescue Inc NEC  
68 Corporation, Network Applications Consortium (NAC), Neustar, Newspaper Association  
69 of America, New Zealand Government State Services Commission, NHK (Japan  
70 Broadcasting Corporation) Science & Technical Research Laboratories, Nippon  
71 Telegraph and Telephone Company, Nokia Corporation, Nortel, NorthID Oy, Norwegian  
72 Agency for Public Management and eGovernment, Norwegian Public Roads  
73 Administration, Novell, NRI Pacific, Office of the Information Privacy Commissioner of  
74 Ontario, Omnibranch, OpenIAM, Oracle USA, Inc., Organisation Internationale pour la  
75 Sécurité des Transactions Électroniques (OISTE), Oslo University, Our New Evolution,  
76 PAM Forum, Parity Communications, Inc., PayPal, Phase2 Technology, Ping Identity  
77 Corporation, Bob Pinheiro, Platinum Solutions, Postsecondary Electronic Standards  
78 Council (PESC), Purdue University, RSA Security, Mary Ruddy, SAFE Bio-pharma,  
79 SanDisk Corporation, Shidler Center for Law, Andrew Shikiar, Signicat AS, Singapore  
80 Institute of Manufacturing Technology, Software & Information Industry Association,  
81 Software Innovation ASA, Sprint Nextel Corporation, Studio Notarile Genghini-SNG,  
82 Sunderland City Council, SUNET, Sun Microsystems, SwissSign AG, Technische  
83 Universitat Berlin, Telefonica S.A., TeleTrusT, TeliaSonera Mobile Networks AB,  
84 TERENA, Thales e-Security, The Boeing Company, The Financial Services  
85 Roundtable/BITS, The Open Group, The University of Chicago as Operator of Argonne  
86 National Laboratory, TRUSTe, tScheme Limited, UNINETT AS, Universidad  
87 Politecnica de Madrid, University of Birmingham, University of Kent, University of  
88 North Carolina at Charlotte, University of Ottawa (TTBE), U.S. Department of Defense,  
89 VeriSign, Vodafone Group Plc, Web Services Competence Center (WSCC), Zenn New  
90 Media

91

92 All rights reserved.

93 Liberty Alliance Project  
94 Licensing Administrator  
95 c/o IEEE-ISTO  
96 445 Hoes Lane  
97 Piscataway, NJ 08855-1331, USA  
98 info@projectliberty.org

99

---

100	<b>Table of Contents</b>	
101	<b>1 Introduction.....</b>	<b>5</b>
102	1.1 Context.....	5
103	1.2 Assumptions.....	6
104	1.3 Excluded Features.....	6
105	<b>2 SOAP Binding .....</b>	<b>7</b>
106	2.1 SOAP Version.....	7
107	2.2 The SOAPAction HTTP Header.....	7
108	2.3 SOAP Fault Messages .....	7
109	<b>3 Messaging-specific Header Blocks .....</b>	<b>8</b>
110	3.1 Overview of Header Blocks.....	8
111	3.2 The <wsa:MessageID> Header Block .....	8
112	3.2.1 <wsa:MessageID> Value Requirements .....	8
113	3.3 The <wsa:RelatesTo> Header Block .....	9
114	3.4 The <wsa:Action> Header Block.....	9
115	3.5 The <sbf:Framework> Header Block .....	9
116	3.6 The <wsa:To> Header Block.....	10
117	3.7 The <wsse:Security> Header Block .....	10
118	3.7.1 Message Authentication and Integrity .....	11
119	3.7.2 Establishing trust in message signature key.....	12
120	3.7.3 Authentication Assertions .....	12
121	3.7.4 Additional Processing Rules for holder-of-key Assertions .....	13
122	<b>4 Overall Processing Rules.....</b>	<b>14</b>
123	4.1 Constructing and sending a SOAP message.....	14
124	4.2 Receiving and processing a SOAP message.....	19
125	<b>5 Security Considerations .....</b>	<b>22</b>
126	<b>6 References.....</b>	<b>23</b>
127		
129		

## 130 1 Introduction

131 Identity-based web services are expected to play an important role in enabling services  
132 that spans organisational borders since they allow IT systems to be connected in a secure,  
133 privacy-respecting and interoperable manner.

134  
135 The present profile is intended to be a basic, scaled-down version of the Liberty ID-WSF  
136 2.0 SOAP Binding Specification [LIB-SOAP] and Security Mechanisms 2.0 ([LIB-SEC]  
137 and [LIB-SAML]). The basic profile adopts mandatory elements from these  
138 specifications such that a Web Service Consumer implementing the profile should be  
139 able to invoke a Web Service Provider implementing the full Liberty SOAP binding (but  
140 not vice versa).

141  
142 In order to keep the profile basic, self-contained<sup>1</sup> and easy to implement without  
143 knowledge on the other Liberty specifications, the profile is *not* a sub-profile of the other  
144 Liberty specifications. Instead, this document profiles the WS-Addressing SOAP Binding  
145 [WSAv1.0-SOAP] and WS-Security [WSS] directly. Thus, mandatory elements and  
146 processing rules from the Liberty SOAP binding are duplicated here and the profile can  
147 thus be read and implemented independently. Other, non-Liberty specifications including  
148 SOAP, WS-Security and WS-Addressing are referenced and not embedded here in order  
149 to keep the profile light-weight. It is believed that many application developers will not  
150 have to implement these specifications from scratch because they are supported in their  
151 development tools, messaging middleware and application servers.

### 152 1.1 Context

153 The following is an example of a usage scenario supported by the profile and which was  
154 used to gather requirements:

- 155 1. A browser user logs in at a Service Provider using normal SAML web SSO  
156 profiles.
- 157 2. The Service Provider needs to invoke a remote identity-based web service at a  
158 Web Service Provider (WSP) on the user's behalf.
- 159 3. The Service Provider exchanges the user's SAML SSO assertion (or  
160 embedded bootstrap token) for an authentication assertion (also called an  
161 identity token<sup>2</sup>) targeted at the WSP, e.g. by contacting a Security Token  
162 Service (STS) or Discovery Service.
- 163 4. The Service Provider (aka Web Service Consumer) invokes the Web Service  
164 Provider using the SOAP binding described in this profile. The request

---

<sup>1</sup> The profile still relies on the WS-\* specifications such as WS-Addressing and WS-Security.

<sup>2</sup> To be exact this profile uses the Liberty term "Authentication assertion" instead of "Identity token" as this term is not defined in a Liberty context..

- 165 includes the authentication assertion in security headers and is signed by the  
166 sender.  
167 5. The Web Service Provider processes the request and responds synchronously.  
168

## 169 1.2 Assumptions

170 The profile builds on the following assumptions:

- 171 • A Web Service Consumer (WSC) needs to invoke a Web Service Provider (WSP)  
172 on behalf of a user / principal by sending a message and receiving synchronously  
173 a response conforming to this profile.
- 174 • The WSC has already access to the WSP's meta data needed for the invocation  
175 (end points, service interface etc.).
- 176 • Both WSC and WSP possess a means of creating signatures that can be verified  
177 by each other; thus they can establish mutual trust in each other's signing key.
- 178 • The WSC has obtained an authentication assertion in the form of an SAML 2.0  
179 assertion which describes the identity of the user whose identity-based web  
180 service is being invoked (invoking identity). The authentication assertion can be  
181 obtained by several means including a Liberty Discovery Service or a STS  
182 implementing the WS-Trust specification.
- 183 • The WSP is able to validate the authentication assertion.  
184

185 These assumptions (along with the excluded features listed below) are the basis for the  
186 formulation of a simplified profile.  
187

## 188 1.3 Excluded Features

189 The following features from [LIB-SOAP] have been excluded in order to formulate a  
190 simpler profile:

- 191 • Endpoint update
- 192 • Processing context header
- 193 • Asynchronous messages
- 194 • Security tokens other than SAML 2.0 assertions
- 195 • Message authentication and -integrity established by other means that signing the  
196 request
- 197 • User interaction
- 198 • Usage directives
- 199 • One user invoking a service on behalf of another user

## 200 **2 SOAP Binding**

### 201 **2.1 SOAP Version**

202 This profile depends upon SOAP version 1.1 as specified in [SOAPv1.1]. Messages  
203 conformant to this specification MUST also be conformant to [SOAPv1.1].

### 204 **2.2 The SOAPAction HTTP Header**

205 [SOAPv1.1] defines the SOAPAction HTTP header, and requires its usage on HTTP-  
206 bound messages.

207

208 The value of the SOAPAction HTTP header SHOULD be the same as the value of the  
209 <wsa:Action> header block defined in the next chapter.

210

### 211 **2.3 SOAP Fault Messages**

212 When reporting a SOAP processing error such as "S:VersionMismatch" or  
213 "S:MustUnderstand", the <S:Fault> element SHOULD be constructed according to  
214 [SOAPv1.1].

215

216 When reporting a WS-Addressing processing error such as "wsa:InvalidAddress", the  
217 <S:Fault> element SHOULD be constructed according to [WSAv1.0-SOAP].

218

219 For all other processing errors the <S:Fault> element's attributes and child elements  
220 MUST be constructed according to these rules:

221

- 222 1. The <S:Fault> element:
  - 223 a. SHOULD contain a <faultcode> element whose value SHOULD be one  
224 of "sbf:FrameworkVersionMismatch", "S:server" or "S:client".
  - 225 b. SHOULD contain a <faultstring> element. This string value MAY be  
226 localized.
  - 227 c. SHOULD NOT contain a <S:faultactor> element.
- 228 2. The <S:Fault> element's <detail> child element SHOULD contain a <Status>  
229 element which:
  - 230 a. MUST contain a code attribute.
  - 231 b. MAY contain a ref attribute.
  - c. MAY contain a comment attribute. This string value MAY be localized.

## 232 **3 Messaging-specific Header Blocks**

233 This section profiles the use of WS-Addressing SOAP Binding [WSAv1.0-SOAP] and  
234 WS-Security [WSS] header blocks, and incorporates the framework header from the  
235 Liberty SOAP Binding [LIB-SOAP].

236

237 Along with header block descriptions are included processing rules the sender must apply  
238 when including it in an outgoing message or when processing it is part of an incoming  
239 message.

240

241 When sending a response to a request, the same header blocks and processing rules apply  
242 unless stated otherwise below. The main difference is that response messages do not  
243 include authentication assertions representing a user.

### 244 **3.1 Overview of Header Blocks**

245 The following header blocks **MUST** be included in the SOAP header:

246

- 247 • <wsa:MessageID>
- 248 • <wsa:RelatesTo> (mandatory on response)
- 249 • <wsa:Action>
- 250 • <wsse:Security>
- 251 • <sbf:Framework>

251

252 The following headers **MAY** be included in the SOAP header:

253 <wsa:To>

254

255 If included, the recipient **SHOULD** be able to process them according to the requirements  
256 described below.

257

258

### 259 **3.2 The <wsa:MessageID> Header Block**

260 The <wsa:MessageID> header block is defined in [WSAv1.0-SOAP]. The value of this  
261 header block uniquely identifies the message that contains it.

262

263 Every message **MUST** contain exactly one such header block.

264

#### 265 **3.2.1 <wsa:MessageID> Value Requirements**

266 Values of the <wsa:MessageID> header block **MUST** satisfy the following property:

267

268 Any party that assigns a value to a <wsa:MessageID> header block **MUST** ensure that  
269 there is negligible probability that ~~that the~~ party or any other party will accidentally  
270 assign the same identifier to any other message.

271  
272 The mechanism by which senders or receivers ensure that an identifier is unique is left to  
273 implementations. In the case that a pseudorandom technique is employed, the above  
274 requirement MAY be met by randomly choosing a value 160 bits in length.

275  
276 Note that [WSAv1.0] requires that `<wsa:MessageID>` values be absolute IRIs.

### 277 **3.3 The `<wsa:RelatesTo>` Header Block**

278 The `<wsa:RelatesTo>` header block is defined in [WSAv1.0-SOAP].

279  
280 The header block MUST be included exactly once in responses to prior-received request  
281 messages. If the `RelationshipType` attribute is included it MUST be set to the value  
282 `http://www.w3.org/2005/03/addressing/reply`.

283  
284 In response messages, the value of this header block MUST be set to the value of the  
285 `<wsa:MessageID>` header block of the prior-received message.

286

### 287 **3.4 The `<wsa:Action>` Header Block**

288 The `<wsa:Action>` header block is defined in [WSAv1.0-SOAP]. The value of this  
289 header block uniquely identifies the semantics implied by the message.

290

291 The header block MUST be included exactly once in all messages.

292

#### 293 **Note**

294 The value of this header block SHOULD contain the same value as the `SOAPAction`  
295 HTTP header defined in [SOAPv1.1]. The SOAP specification requires the HTTP header  
296 on all HTTP-bound SOAP messages.

297

298

### 299 **3.5 The `<sbef:Framework>` Header Block**

300 The `<sbef:Framework>` header block is defined in the [LIB-SOAP] specification and  
301 provides the sender with a means to communicate the version of the ID-WSF framework  
302 used to construct the message. In order to make messages produced using this profile  
303 compatible with the full Liberty SOAP binding, the Liberty framework header is used in  
304 this profile as well.

305

306 The header block MUST be included exactly once in every message.

307

308 Further:

309 The `version` attribute SHOULD be set to “2.0”

310

311 A `profile` attribute with the name space “`urn:liberty:sb:profile`” MUST be  
312 included with the value of “`urn:liberty:sb:profile:basic`”.

313

314

315 Example:

316

```
317 <sbf:Framework  
318     xmlns:sbfprofile="urn:liberty:sb:profile"  
319     ...  
320     version="2.0"  
321     sbfprofile:profile="urn:liberty:sb:profile:basic"  
322     s:mustUnderstand="1"  
323     s:actor="http://schemas.../next"  
324     wsu:Id="SBF"/>
```

325

326 If the receiver of a message does not recognize the `version` and `profile` attributes, it  
327 MAY respond to the sender with a SOAP fault message with the `<faultcode>` of  
328 `sbf:FrameworkVersionMismatch`.

329

### 330 3.6 The `<wsa:To>` Header Block

331 The `<wsa:To>` header block is defined in [WSAv1.0-SOAP]. The value of this header  
332 block specifies the intended destination of the message.

333

#### 334 **Note**

335 In the typical case that a WS-Addressing endpoint reference is used to address a message,  
336 the value of this header block is taken from the `<wsa:Address>` of the endpoint  
337 reference. If the `<wsa:To>` header block is not present, the value defaults to  
338 `http://www.w3.org/2005/03/addressing/role/anonymous`; so, when constructing a  
339 message, the header block can be omitted if this is the value that would be used. This  
340 typically allows the `<wsa:To>` header block to be omitted in responses during  
341 synchronous request-response message exchanges over HTTP.

342

343 The header block is optional.

### 344 3.7 The `<wsse:Security>` Header Block

345 This section defines elements and processing rules for SOAP message security by  
346 profiling the `<wsse:Security>` header block defined in [WSS]. Processing rules defined  
347 in [WSS] and [WSS-STP] MUST be followed unless stated explicitly otherwise below.

348

349 A single `<wsse:Security>` header block MUST be present and MUST have a  
350 `mustUnderstand` attribute with the logical value of `true`. Further, it MUST include a  
351 `<wsu:Timestamp>` with a `<wsu:Created>` element.  
352

353 The value of the `<wsu:Created>` element SHOULD be within an appropriate offset from  
354 local time. Absent other guidance, a value of 5 minutes MAY be used.  
355

356 If the `<wsu:Timestamp>` element includes an `<wsu:Expires>` element, the receiver  
357 MUST ensure that his local time is before that time.  
358

359 To prevent message replay, receivers SHOULD maintain a message cache, and check  
360 received `messageID` values against the cache. How long time a message should be kept in  
361 the cache at the WSP is governed by deployment policy.  
362  
363  
364

### 365 3.7.1 Message Authentication and Integrity

366 Authentication and integrity of messages is established by means of digital signatures  
367 applied to the SOAP message. Confidentiality, if required, MUST be established by using  
368 a secure transport protocol (e.g. using SSL 3.0 or TLS 1.1 or later).  
369

370 The sender MUST create and include a single `<ds:Signature>` element in the  
371 `<wsse:Security>` header block and this signature MUST reference:

- 372 • The SOAP `<Body>` element
- 373 • All security tokens embedded directly under the `<wsse:Security>` element via a  
374 `<wsse:SecurityTokenReference>` (see below), and
- 375 • All SOAP header blocks in the message defined in this profile. The signature  
376 MAY reference other elements including header blocks not mentioned in this  
377 profile.  
378

379  
380 If the sender has obtained a SAML holder-of-key Assertion vouching for the signing key  
381 (see next section) it SHOULD be included in the security header. Detailed requirements  
382 for using holder-of-key assertions are given below.  
383

384 If the sender does not possess a holder-of-key Assertion but instead has an X.509  
385 certificate, the certificate SHOULD be included in a `<wsse:BinarySecurityToken>`  
386 element in the security header. In the message signature, the `<ds:KeyInfo>` element  
387 SHOULD refer to this token via a `<wsse:SecurityTokenReference>`.  
388

389 The receiver MUST validate the message signature and security tokens including test of  
390 validity period and trust in the token issuer. Depending on local policy, the receiver

391 SHOULD check revocation status of any certificates used to sign the message and  
392 tokens.  
393

### 394 3.7.2 Establishing trust in message signature key

395 The receiver can establish trust in the sender's signature key in the following ways:

- 396 • The security header contains a SAML 2.0 holder-of-key assertion issued by  
397 someone<sup>3</sup> the receiver trusts, and the holder-of-key assertion includes a key that  
398 can be used to verify the message signature. Note that the assertion itself will be  
399 signed by the trusted issuer so the receiver has to be able to verify the issuer's  
400 signature. The sender's signing key MAY be symmetric or asymmetric.
- 401 • The message is signed with a key the receiver already knows / trusts for example  
402 due to prior metadata exchange.
- 403 • The security header includes an X.509 certificate in a BinarySecurityToken  
404 issued by a Certificate Authority the receiver trusts, and the certificate can be  
405 used to verify the message signature.  
406

### 407 3.7.3 Authentication Assertions

408 In request messages, the <wsse:Security> header block MAY include authentication  
409 assertions in the form of SAML 2.0 assertions representing the identity of the user /  
410 principal whose identity-based web service is being invoked. Other types of security  
411 tokens (except for BinarySecurityTokens containing certificates) SHOULD not be used  
412 and implementations of this profile are not required to implement them.  
413

414 The authentication assertion MUST be a SAML 2.0 assertion with subject confirmation  
415 method being either urn:oasis:names:tc:SAML:2.0:cm:bearer OR  
416 urn:oasis:names:tc:SAML:2.0:cm:holder-of-key.  
417

418 Authentication assertions MUST be signed by the issuer (e.g. Identity Provider, STS or  
419 Discovery Service). Requirements for the content of authentication assertions are not  
420 specified further in this profile.  
421

422 | Authentication assertions MUST be signed by the sender [by](#) including first a  
423 <wsse:SecurityTokenReference> in <wsse:Security> header block, and then  
424 referencing the STR from the message signature using a <ds:Reference> element. The  
425 security token reference MUST include a <wsse:KeyIdentifier> with a ValueType of  
426 http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID  
427 and specify the ID of the SAML assertion. The <ds:Reference> element MUST use a  
428 transform algorithm set to "http://docs.oasis-open.org/wss/2004/01/oasis-  
429 200401-wsssoap-message-security-1.0#STR-Transform".

---

<sup>3</sup> For example a Liberty Discovery Service or a Security Token Service.

430  
431 The receiver MUST validate SAML 2.0 authentication assertions according to the  
432 processing rules defined in [SAML-CORE] and [WSS-STP] including life time of the  
433 token, audience restriction, the issuer's signature over the token, trust in the issuer and  
434 other processing rules defined by token profiles.  
435

### 436 **3.7.4 Additional Processing Rules for holder-of-key Assertions**

437 When the authentication assertion has a subject confirmation method being "holder-of-  
438 key" it means that the sender must prove possession of a key mentioned in the assertion's  
439 <SubjectConfirmationData> in order for the recipient to rely on the assertion. The  
440 proof-of-possession of the key will be achieved via the message signature and provides  
441 additional assurance that the sender is allowed to use to the assertion in a web service  
442 invocation.  
443

444 In this profile, a holder-of-key Assertion MUST in the <SubjectConfirmationData>  
445 element include a key that can be used to verify the message signature. Thus, the *same*  
446 key used for message authentication and integrity is used to confirm the right to use the  
447 assertion for message authorization purposes.  
448

449 The message signature (i.e. the <ds:Signature> element) MUST refer to the token with  
450 the subject confirmation key within the <ds:KeyInfo> element.  
451

452 The receiver MUST check that the message is signed by same key mentioned in the  
453 assertion's subject confirmation element before relying on the assertion content.

## 454 **4 Overall Processing Rules**

455 Overall processing of SOAP-bound messages follows the rules of the SOAP processing  
456 model described in [SOAPv1.1]. A number of additional rules are defined below. Notice  
457 that processing rules for individual elements are found in the previous section.  
458

### 459 **4.1 Constructing and sending a SOAP message**

460 The sender **MUST** follow these processing rules when constructing and sending an  
461 outgoing SOAP message:

- 462
- 463 1. The outgoing message **MUST** satisfy the rules for SOAP binding defined in  
464 section “SOAP Binding”.
- 465 2. The outgoing message **MUST** satisfy the rules for WS-Addressing SOAP binding  
466 given in [WSAv1.0-SOAP].
- 467 3. The outgoing message **MUST** include the mandatory header blocks defined  
468 above.
- 469 4. All other Liberty headers defined in [LIB-SOAP] **SHOULD NOT** be used with  
470 this profile since implementations of the profile are not required to support them.
- 471 5. Each header block included in the outgoing message **MUST** conform to the  
472 processing rules defined for each header block.
- 473

474 Below is shown a procedure that illustrates how a conforming message can be  
475 constructed (some low-level details have been omitted). It is assumed that the sender has  
476 obtained all the information required to construct the message including security tokens,  
477 signing keys and message payload. The procedure is not normative and conforming  
478 messages can be constructed in other ways:

- 479
- 480 1. Construct the XML payload to be included in the SOAP Body.
- 481 2. Construct a SOAP envelope with <Header> and <Body>, and embed the payload  
482 in the <Body>. Add a `wsu:Id` attribute<sup>4</sup> to the <Body> element.
- 483 3. Add a <wsa:MessageID> header block (including a `wsu:Id` attribute) which  
484 uniquely identifies the message; for example generate a 160-bit pseudorandom  
485 number and embed it in a URI as follows:  
486  
487 `http://spwsp.com/ffeeddccbaa99887766 554433221100ffeebbcc`  
488
- 489 4. When generating a response, include a <wsa:RelatesTo> element (including a  
490 `wsu:Id` attribute) containing the message ID of the request.

---

<sup>4</sup> In the following, all `wsu:Id` attributes should contain a value that is unique within the SOAP message.

- 491 5. Add a <wsa:Action> header block (including a wsu:Id attribute) corresponding  
492 to the SOAPAction HTTP header as required by the service being invoked.
- 493 6. If required, add a <wsa:To> header block (including a wsu:Id attribute) to  
494 identify the recipient.
- 495 7. Add the <sbef:Framework> header block as defined previously (including a  
496 wsu:Id attribute).
- 497 8. Add a <wsse:Security> header block with a mustUnderstand=1 attribute.
  - 498 a. Add a <wsu:Timestamp> element (including a wsu:Id attribute) with a  
499 <wsu:Created> sub-element that includes the local time.
  - 500 b. Include any security tokens (SAML Assertions and/or  
501 BinarySecurityTokens containing X.509 certificates) in the security  
502 header block. Ensure that they have unique id attributes so they can be  
503 referenced (e.g. saml2:ID or wsu:Id).
  - 504 c. Create a <wsse:SecurityTokenReference> element (including a wsu:Id  
505 attribute) for each embedded SAML assertion. Add a TokenType attribute  
506 stating the type of token (<http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0>) and a  
507 <wsse:KeyIdentifier> sub-element containing the ID of the assertion.
  - 508 d. Create a <ds:Signature> element in the security header:
    - 509 i. Add a <ds:SignedInfo> element and embed <ds:Reference>  
510 sub-elements with references to each of the above header blocks  
511 and the SOAP Body. For each reference, include element ID,  
512 digest method and digest value. Set the Transform Algorithm to  
513 <http://www.w3.org/2001/10/xml-exc-c14n#>  
514
    - 515 ii. Include a <ds:Reference> elements for each assertion reference  
516 produced in step c) by using the ID of the  
517 <SecurityTokenReference> element. Set the Transform  
518 Algorithm set to [http://docs.oasis-](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsssoap-message-security-1.0#STR-Transform)  
519 [open.org/wss/2004/01/oasis-200401-wsssoap-message-](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsssoap-message-security-1.0#STR-Transform)  
520 [security-1.0#STR-Transform](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsssoap-message-security-1.0#STR-Transform)
  - 521 e. Add a <ds:KeyInfo> element with a <wsse:SecurityTokenReference>  
522 pointing to either a SAML assertion or BinarySecurityToken vouching for  
523 the signature key. The reference should include a <wsse:KeyIdentifier>  
524 containing the ID of the token.
  - 525 f. Compute the <ds:SignatureValue> over the <ds:SignedInfo> using  
526 the signature key.
- 527 9. Send the message over a secure transport (SSL or TLS).

530 Below is shown an example SOAP message that is compliant with the Liberty Basic  
531 SOAP binding:  
532

```
533 <?xml version="1.0" encoding="UTF-8"?>
534 <s:Envelope
535     xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
536     xmlns:sbf="urn:liberty:sb"
537     xmlns:sbfprofile="urn:liberty:sb:profile"
538     xmlns:sec="urn:liberty:security:2006-08"
539     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
540         1.0.xsd"
541     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
542         1.0.xsd"
543     xmlns:wsa="http://www.w3.org/2005/08/addressing"
544     xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
545
546     <s:Header>
547         <wsa:MessageID wsu:Id="mid">f63d289c-cd9a-4c00-bf87-c4bad0310646</wsa:MessageID>
548
549         <wsa:To wsu:Id="to">...</wsa:To>
550
551         <wsa:Action wsu:Id="action">urn:liberty:id-sis-pp:2003-08:Modify</wsa:Action>
552
553
554     <sbf:Framework
555         version="2.0"
556         sbfprofile:profile="urn:liberty:sb:profile:basic"
557         s:mustUnderstand="1"
558         s:actor="http://schemas.../next"
559         wsu:Id="framework"/>
560
561
562     <wsse:Security mustUnderstand="1">
563         <wsu:Timestamp wsu:Id="ts">
564             <wsu:Created>2008-08-17T04:49:17Z</wsu:Created >
565         </wsu:Timestamp>
566
567         <!-- this is the holder-of-key token with the sender's certificate -->
568         <saml2:Assertion
569             xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
570             Version="2.0"
571             ID="sxJu9g/vvLG9sAN9bKp/8q0NKU="
572             IssueInstant="2008-08-01T16:58:33Z">
573             <saml2:Issuer>http://authority.example.com/</Saml2:Issuer>
574
575             <!-- signature by the issuer over the assertion -->
576             <ds:Signature>
577                 <ds:SignedInfo>
578                     <ds:CanonicalizationMethod
579                         Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
580                     <ds:SignatureMethod
581                         Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
582                     <ds:Reference URI="#sxJu9g/vvLG9sAN9bKp/8q0NKU=">
583                         <ds:Transforms>
584                             <ds:Transform
585                                 Algorithm="http://www.w3.org/2000/09/xmldsig#envelopedsignature" />
586                         </ds:Transforms>
```

```
587         <ds:DigestMethod
588             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
589
590         <ds:DigestValue>TCDVSuG6grhyHbzhQFWFzGrxIPE=</ds:DigestValue>
591     </ds:Reference>
592 </ds:SignedInfo>
593 <ds:SignatureValue>
594 x/GyPbzmFEe85pGD3c1aXG4Vspb9V9jGCjwcRCKrtwPS6vdVNCcY5rHaFPYWkf+5
595 EIYcPzx+pX1h43SmwviCqXRjRtMANWbHLhWAptaK1yWS7gFgsD01qjyen3CP+m3D
596 w6vKhaqledl0BYyrIzb4KkHO4ahNyBVXbJwqv5pUaE4=
597 </ds:SignatureValue>
598 <ds:KeyInfo>
599     <ds:X509Data>
600     <!-- data identifying the signer's certificate -->
601     </ds:X509Data>
602 </ds:KeyInfo>
603 </ds:Signature>
604
605 <saml2:Subject>
606     <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">
607         005a06e0-ad82-110d-a556-004005b13a2b
608     </saml2:NameID>
609
610     <!-- Here comes the subject confirmation method saying this is a holder-of-
611 key -->
612     <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-
613 of-key">
614
615         <!-- Here comes a NameID indicating the ID of the sender who must
616 confirm with a key -->
617         <saml2:NameID format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
618             http://wsc.someorg.com
619         </saml2:NameID>
620
621         <!-- Here comes info on the key to confirm with (same as signing key) --
622 >
623         <saml2:SubjectConfirmationData
624 xsi:type="saml2:KeyInfoConfirmationDataType"
625         <ds:KeyInfo>
626         <ds:X509Data>
627         <!-- Here comes the sender's X509 cert -->
628             MIIB9zCCAWSgAwIBAgIQ...
629         </ds:X509Data>
630         </ds:KeyInfo>
631         </saml2:SubjectConfirmationData>
632
633     </saml2:SubjectConfirmation>
634 </saml2:Subject>
635
636     <!-- Entity which should consume the information in the assertion. -->
637 <saml2:Conditions
```

```
641         NotOnOrAfter="2008-08-01T21:42:43Z">
642         <saml2:AudienceRestrictionCondition>
643             <saml2:Audience>http://wsp.example.com</saml2:Audience>
644         </saml2:AudienceRestrictionCondition>
645     </saml2:Conditions>
646
647     <saml2:AttributeStatement>
648         ...
649     </saml2:AttributeStatement>
650 </saml2:Assertion>
651
652     <!-- This SecurityTokenReference is used to reference the SAML Assertion from a
653 ds:Reference -->
654     <wsse:SecurityTokenReference
655         xmlns:wsse="..." xmlns:wsu="..." xmlns:wssell="..."
656         wsu:Id="str1"
657         wssell:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
658 1.1#SAMLV2.0">
659     <!-- A key identifier with the SAML Assertion ID -->
660     <wsse:KeyIdentifier
661         ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
662 1.1#SAMLID">
663         sxJu9g/vvLG9sAN9bKp/8q0NKU=
664     </wsse:KeyIdentifier>
665 </wsse:SecurityTokenReference>
666
667
668     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
669         <ds:SignedInfo>
670             <!-- in general include a ds:Reference for each wsa: header added according
671 to SOAP binding -->
672
673             <!-- include the MessageID in the signature -->
674             <ds:Reference URI="#mid">...</ds:Reference>
675
676             <!-- include the To in the signature -->
677             <ds:Reference URI="#to">...</ds:Reference>
678
679             <!-- include the Action in the signature -->
680             <ds:Reference URI="#action">...</ds:Reference>
681
682             <!-- include the Framework in the signature -->
683             <ds:Reference URI="#framework">...</ds:Reference>
684
685             <!-- include the Timestamp in the signature -->
686             <ds:Reference URI="#ts">...</ds:Reference>
687
688             <!-- include the SAML Assertion in the signature to avoid token
689 substitution attacks -->
690             <ds:Reference URI="#str1">
691                 <ds:Transform Algorithm="http://docs.oasis-open.org/wss/2004/01/oasis-
692 200401-wsssoap-message-security-1.0#STR-Transform">
693                     <wsse:TransformationParameters>
694                         <ds:CanonicalizationMethod
```

```
695         Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
696         </wsse:TransformationParameters>
697     </ds:Transform>
698 </ds:Reference>
699
700     <!-- bind the body of the message -->
701     <ds:Reference URI="#MsgBody">
702         <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
703         <ds:DigestValue>YgGfS0pi56pu...</ds:DigestValue>
704     </ds:Reference>
705 </ds:SignedInfo>
706
707 <!-- include a security token reference for holder-of-key confirmation -->
708 <ds:KeyInfo>
709     <wsse:SecurityTokenReference
710         xmlns:wsse="..." xmlns:wsu="..." xmlns:wssell="..."
711         wsu:Id="str2"
712         wssell:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-
713 profile-1.1#SAMLV2.0">
714         <!-- A key identifier with the SAML Assertion ID -->
715         <wsse:KeyIdentifier
716             ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-
717 profile-1.1#SAMLID">
718             sxJu9g/vvLG9sAN9bKp/8q0NKU=
719         </wsse:KeyIdentifier>
720     </wsse:SecurityTokenReference>
721 </ds:KeyInfo>
722
723     <ds:SignatureValue>
724         HJJWbvqW9E84vJVQkjLLA6nNvBX7mY00TzhwBdFNDElgscSXZ5Ekw==
725     </ds:SignatureValue>
726 </ds:Signature>
727 </wsse:Security>
728 </s:Header>
729
730 <s:Body wsu:Id="MsgBody">
731     <idpp:Modify>
732         : <!-- this is an ID-PP Modify message -->
733     </idpp:Modify> </s:Body>
734 </s:Envelope>
```

## 735 4.2 Receiving and processing a SOAP message

736 The receiver of a SOAP message (either normal message or fault) MUST perform the  
737 following tests on the header blocks:

738

739 Note: Although the steps are numbered sequentially, implementations MAY use a  
740 different sequence as long as all tests are applied.

741

- 742 1. The incoming message MUST satisfy the rules for SOAP binding defined in  
743 section "SOAP Binding".
- 744 2. The incoming message MUST satisfy the rules given in [WSAv1.0-SOAP].

- 745 3. The incoming message MUST include all mandatory header blocks defined  
746 above.  
747 4. Each header block in the message (mandatory as well as optional) MUST be  
748 tested according to the processing rules defined above.  
749

750 Below is shown a procedure illustrating how messages can be verified and processed  
751 (some details e.g. regarding signature processing have been omitted; for details see the  
752 XML digital signature standard). It is assumed that the receiver has all the information  
753 required to process the message including certificates of trusted parties issuing tokens.  
754 The procedure is not normative and messages may be processed / validated in other  
755 ways; implementations may for example perform the steps in other sequence for  
756 efficiency reasons.

- 757
- 758 1. Receive the SOAP message over a secure transport protocol (SSL or TLS).
  - 759 2. Validate that the following mandatory SOAP headers are present and contain  
760 appropriate values: `<wsa:MessageID>` should include a unique value,  
761 `<sbef:Framework>` should specify a framework version and profile understood by  
762 the recipient and `<wsa:Action>` should be consistent with the invoked service.
  - 763 3. If present, check that the content of the `<wsa:To>` header corresponds to the  
764 recipient / endpoint.
  - 765 4. Check the received message ID value against the local cache to determine  
766 whether it has been received before (replay attacks). If not, add message ID to  
767 cache to detect future replays.
  - 768 5. Check that exactly one `<wsse:Security>` header is present:
    - 769 a. Verify that the `<wsu:Timestamp>` is within acceptable limits of local  
770 server time as defined by deployment policy.
    - 771 b. Validate all embedded security tokens including that they are signed by a  
772 trusted issuer, timestamps, audience restrictions etc. (token validation  
773 rules vary with token type). Any proof-of-possession requirements are  
774 handled below.
    - 775 c. Check that the message signature (`<ds:Signature>`) contains references  
776 to all header block defined above, to the SOAP body and all included  
777 SAML assertions (via a `SecurityTokenReference`). Verify that all digest  
778 values match the referenced elements.
    - 779 d. Verify the message signature using the key referenced in the  
780 `<ds:KeyInfo>` element.
    - 781 e. Check that the signing key is vouched-for via a security token issued by a  
782 trusted party.
    - 783 f. Verify that proof-of-possession requirements in tokens (e.g. SAML  
784 holder-of-key `SubjectConfirmation`) are demonstrated via the message  
785 signing key. Thus, the proof-of-possession key in tokens must match the  
786 key that signed the message.

787                   g. Check that all claims required by the service have been demonstrated by  
788                   the attached security tokens.

789           6. Discard message payload if any of the above checks fail and send a meaningful  
790           error message to the recipient.

791           7. Handle message payload and send response over secure transport.

792

793   Note that the recipient may need to perform additional checks e.g. related to  
794   authorization.

795

## 796 5 Security Considerations

797 Message integrity and authenticity is established by mandatory signing (and subsequent  
798 verification) of the SOAP body, header blocks in this specification and security tokens.  
799

800 Message confidentiality is not addressed directly in this profile but may be established by  
801 using a secure transport protocol such as SSL 3.0, TLS 1.1 or later HTTPS, or by  
802 encryption of name identifiers or individual attributes in the SAML 2.0 assertion.  
803

804 Message freshness and prevention against replay attacks is established by including  
805 unique message Ids that WSP's should cache, ~~and~~ time stamps ~~as well as~~ expiry of  
806 tokens. How long time a message should be kept in the cache at the WSP is governed by  
807 deployment policy.  
808

809 Message authorization is established by including signed authentication assertions in the  
810 form of SAML assertions issued by a trusted STS, Liberty Discovery Service or Identity  
811 Provider.  
812

813 Security tokens in the form of SAML 2.0 assertions are signed by the issuer and sensitive  
814 attributes may be encrypted if deemed necessary via the mechanisms described in  
815 [SAML-CORE] including encryption of the entire assertion, name identifiers and  
816 individual attributes.  
817

818 It is outside the scope of this profile to define how a Web Service Provider performs local  
819 authorization decisions but the WSP may take the following request parameters into  
820 consideration:

- 821 • The sender identity as established via the signature.
- 822 • The invoker / user identity as established via authentication assertions.
- 823 • The resource / service being accessed.
- 824 • Trust in the STS, Discovery Service or Identity Provider that has issued the  
825 authentication assertion.
- 826 • The assurance level established as part of the assertion.

827 **6 References**

- [SOAPv1.1] “Simple Object Access Protocol (SOAP) 1.1,” Box, Don, Ehnebuske et. al. World Wide Web Consortium W3C Note (08 May 2000).  
<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>”.
- [WSS] “Web Services Security: SOAP Message Security 1.1”, OASIS Standard, 1 February 2006.
- [WSS-STP] “Web Services Security: SAML Token Profile 1.1”, OASIS Standard, 1 February 2006.  
<http://docs.oasis-open.org/wss/oasis-wss-SAMLTOKENProfile-1.1>
- [SAML-CORE] “Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0”, OASIS Standard, 15 March 2005.
- [LIB-SOAP] “Liberty ID-WSF SOAP Binding Specification”, version 2.0, Liberty Alliance Project
- [LIB-SAML] “ID-WSF 2.0 SecMech SAML Profile”, version 2.0, Liberty Alliance Project.
- [LIB-SEC] “Liberty ID-WSF Security Mechanisms Core”, version 2.0, Liberty Alliance Project.
- [WSS-SAML] “Web Services Security: SAML Token Profile 1.1”, OASIS Standard, 1 February 2006.
- [Scenarios] “Identity-Based Web Services – Scenarios”, Danish IT and Telecom Agency. (Not yet published on the WWW)
- [WSAv1.0-SOAP] “WS-Addressing 1.0 SOAP Binding”, World Wide Web Consortium W3C Recommendation (9 May 2006).

828