

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, Sym Labs
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_Final.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 hereby
24 granted to use the document solely for the purpose of implementing the Specification. No
25 rights are granted to prepare derivative works of this Specification. Entities seeking
26 permission to reproduce portions of this document for other uses must contact the Liberty
27 Alliance to determine whether an appropriate license for such use is available.

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

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

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

87

88 All rights reserved.

89 Liberty Alliance Project
90 Licensing Administrator
91 c/o IEEE-ISTO
92 445 Hoes Lane
93 Piscataway, NJ 08855-1331, USA
94 info@projectliberty.org

95

96	Table of Contents	
97	1 Introduction	5
98	1.1 Context	5
99	1.2 Assumptions	6
100	1.3 Excluded Features	6
101	2 SOAP Binding	7
102	2.1 SOAP Version	7
103	2.2 The SOAPAction HTTP Header	7
104	2.3 SOAP Fault Messages	7
105	3 Messaging-specific Header Blocks	8
106	3.1 Overview of Header Blocks	8
107	3.2 The <wsa:MessageID> Header Block	8
108	3.2.1 <wsa:MessageID> Value Requirements	8
109	3.3 The <wsa:RelatesTo> Header Block	9
110	3.4 The <wsa:Action> Header Block	9
111	3.5 The <sbfc:Framework> Header Block	9
112	3.6 The <wsa:To> Header Block	10
113	3.7 The <wsse:Security> Header Block	10
114	3.7.1 Message Authentication and Integrity	11
115	3.7.2 Establishing trust in message signature key	11
116	3.7.3 Authentication Assertions	12
117	3.7.4 Additional Processing Rules for holder-of-key Assertions	13
118	4 Overall Processing Rules	14
119	4.1 Constructing and sending a SOAP message	14
120	4.2 Receiving and processing a SOAP message	19
121	5 Security Considerations	22
122	6 References	23
123		
125		

126 1 Introduction

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

130

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

136

137 In order to keep the profile basic, self-contained¹ and easy to implement without knowledge
138 on the other Liberty specifications, the profile is *not* a sub-profile of the other Liberty
139 specifications. Instead, this document profiles the WS-Addressing SOAP Binding
140 [WSAv1.0-SOAP] and WS-Security [WSS] directly. Thus, mandatory elements and
141 processing rules from the Liberty SOAP binding are duplicated here and the profile can thus
142 be read and implemented independently. Other, non-Liberty specifications including SOAP,
143 WS-Security and WS-Addressing are referenced and not embedded here in order to keep the
144 profile light-weight. It is believed that many application developers will not have to
145 implement these specifications from scratch because they are supported in their development
146 tools, messaging middleware and application servers.

147 1.1 Context

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

- 150 1. A browser user logs in at a Service Provider using normal SAML web SSO
151 profiles.
- 152 2. The Service Provider needs to invoke a remote identity-based web service at a
153 Web Service Provider (WSP) on the user's behalf.
- 154 3. The Service Provider exchanges the user's SAML SSO assertion (or embedded
155 bootstrap token) for an authentication assertion (also called an identity token²)
156 targeted at the WSP, e.g. by contacting a Security Token Service (STS) or
157 Discovery Service.
- 158 4. The Service Provider (aka Web Service Consumer) invokes the Web Service
159 Provider using the SOAP binding described in this profile. The request includes
160 the authentication assertion in security headers and is signed by the sender.
- 161 5. The Web Service Provider processes the request and responds synchronously.

¹ The profile still relies on the WS-* specifications such as WS-Addressing and WS-Security.

² 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..

162

163 1.2 Assumptions

164 The profile builds on the following assumptions:

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

178

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

181

182 1.3 Excluded Features

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

- 185 • Endpoint update
- 186 • Processing context header
- 187 • Asynchronous messages
- 188 • Security tokens other than SAML 2.0 assertions
- 189 • Message authentication and -integrity established by other means than signing the
190 request
- 191 • User interaction
- 192 • Usage directives
- 193 • One user invoking a service on behalf of another user

194 2 SOAP Binding

195 2.1 SOAP Version

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

198 2.2 The SOAPAction HTTP Header

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

201
202 The value of the SOAPAction HTTP header SHOULD be the same as the value of the
203 <wsa:Action> header block defined in the next chapter.

204

205 2.3 SOAP Fault Messages

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

209

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

212

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

215

1. The <S:Fault> element:

216

a. SHOULD contain a <faultcode> element whose value SHOULD be one of
217 "sbf:FrameworkVersionMismatch", "s:server" or "s:client".

218

b. SHOULD contain a <faultstring> element. This string value MAY be
219 localized.

220

c. SHOULD NOT contain a <S:faultactor> element.

221

2. The <S:Fault> element's <detail> child element SHOULD contain a <Status>
222 element which:

223

a. MUST contain a code attribute.

224

b. MAY contain a ref attribute.

225

c. MAY contain a comment attribute. This string value MAY be localized.

226 **3 Messaging-specific Header Blocks**

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

230

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

234

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

238 **3.1 Overview of Header Blocks**

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

240

- <wsa:MessageID>
- <wsa:RelatesTo> (mandatory on response)
- <wsa:Action>
- <wsse:Security>
- <sbef:Framework>

245

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

247

<wsa:To>

248

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

251

252

253 **3.2 The <wsa:MessageID> Header Block**

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

256

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

258

259 **3.2.1 <wsa:MessageID> Value Requirements**

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

261

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

265
266 The mechanism by which senders or receivers ensure that an identifier is unique is left to
267 implementations. In the case that a pseudorandom technique is employed, the above
268 requirement MAY be met by randomly choosing a value 160 bits in length.

269
270 Note that [WSAv1.0] requires that `<wsa:MessageID>` values be absolute IRIs.

271 **3.3 The `<wsa:RelatesTo>` Header Block**

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

273

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

277

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

280

281 **3.4 The `<wsa:Action>` Header Block**

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

284

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

286

287 **Note**

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

291

292

293 **3.5 The `<sbef:Framework>` Header Block**

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

299

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

301

302 Further:

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

304
305 A profile attribute with the name space “urn:liberty:sb:profile” MUST be
306 included with the value of “urn:liberty:sb:profile:basic”.

307
308

309 Example:

```
310 <sbfc:Framework
311     xmlns:sbfcprofile="urn:liberty:sb:profile"
312     ...
313     version="2.0"
314     sbfcprofile:profile="urn:liberty:sb:profile:basic"
315     s:mustUnderstand="1"
316     s:actor="http://schemas.../next"
317     wsu:Id="SBF"/>
```

319

320 If the receiver of a message does not recognize the `version` and `profile` attributes, it
321 MAY respond to the sender with a SOAP fault message with the `<faultcode>` of
322 `sbfc:FrameworkVersionMismatch`.

323

324 3.6 The `<wsa:To>` Header Block

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

327

328 Note

329 In the typical case that a WS-Addressing endpoint reference is used to address a message, the
330 value of this header block is taken from the `<wsa:Address>` of the endpoint reference. If the
331 `<wsa:To>` header block is not present, the value defaults to

332 `http://www.w3.org/2005/03/addressing/role/anonymous`; so, when constructing a
333 message, the header block can be omitted if this is the value that would be used. This
334 typically allows the `<wsa:To>` header block to be omitted in responses during synchronous
335 request-response message exchanges over HTTP.

336

337 The header block is optional.

338 3.7 The `<wsse:Security>` Header Block

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

342

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

346

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

349

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

352

353 To prevent message replay, receivers SHOULD maintain a message cache, and check
354 received `messageID` values against the cache. How long time a message should be kept in
355 the cache at the WSP is governed by deployment policy.

356

357

358

359 **3.7.1 Message Authentication and Integrity**

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

363

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

366

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

369

370

371

372

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

376

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

381

382 The receiver MUST validate the message signature and security tokens including test of
383 validity period and trust in the token issuer. Depending on local policy, the receiver
384 SHOULD check revocation status of any certificates used to sign the message and tokens.

385

386 **3.7.2 Establishing trust in message signature key**

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

- 388
- 389
- 390
- 391
- 392
- 393
- 394
- 395
- 396
- 397
- 398
- The security header contains a SAML 2.0 holder-of-key assertion issued by someone³ the receiver trusts, and the holder-of-key assertion includes a key that can be used to verify the message signature. Note that the assertion itself will be signed by the trusted issuer so the receiver has to be able to verify the issuer's signature. The sender's signing key MAY be symmetric or asymmetric.
 - The message is signed with a key the receiver already knows / trusts for example due to prior metadata exchange.
 - The security header includes an X.509 certificate in a BinarySecurityToken issued by a Certificate Authority the receiver trusts, and the certificate can be used to verify the message signature.

399 3.7.3 Authentication Assertions

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

405

406 The authentication assertion MUST be a SAML 2.0 assertion with subject confirmation
407 method being either `urn:oasis:names:tc:SAML:2.0:cm:bearer` or
408 `urn:oasis:names:tc:SAML:2.0:cm:holder-of-key`.

409

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

413

414 Authentication assertions MUST be signed by the sender by including first a
415 `<wsse:SecurityTokenReference>` in `<wsse:Security>` header block, and then
416 referencing the STR from the message signature using a `<ds:Reference>` element. The
417 security token reference MUST include a `<wsse:KeyIdentifier>` with a `ValueType` of
418 `http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID`
419 and specify the ID of the SAML assertion. The `<ds:Reference>` element MUST use a
420 transform algorithm set to `"http://docs.oasis-open.org/wss/2004/01/oasis-`
421 `200401-wsssoap-message-security-1.0#STR-Transform"`.

422

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

427

³ For example a Liberty Discovery Service or a Security Token Service.

428 3.7.4 Additional Processing Rules for `holder-of-key` Assertions

429 When the authentication assertion has a subject confirmation method being “`holder-of-`
430 `key`” it means that the sender must prove possession of a key mentioned in the assertion’s
431 `<SubjectConfirmationData>` in order for the recipient to rely on the assertion. The proof-
432 of-possession of the key will be achieved via the message signature and provides additional
433 assurance that the sender is allowed to use to the assertion in a web service invocation.
434

435 In this profile, a `holder-of-key` Assertion **MUST** in the `<SubjectConfirmationData>`
436 element include a key that can be used to verify the message signature. Thus, the *same* key
437 used for message authentication and integrity is used to confirm the right to use the assertion
438 for message authorization purposes.
439

440 The message signature (i.e. the `<ds:Signature>` element) **MUST** refer to the token with
441 the subject confirmation key within the `<ds:KeyInfo>` element.
442

443 The receiver **MUST** check that the message is signed by same key mentioned in the
444 assertion’s subject confirmation element before relying on the assertion content.

445 4 Overall Processing Rules

446 Overall processing of SOAP-bound messages follows the rules of the SOAP processing
447 model described in [SOAPv1.1]. A number of additional rules are defined below. Notice that
448 processing rules for individual elements are found in the previous section.
449

450 4.1 Constructing and sending a SOAP message

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

- 453
- 454 1. The outgoing message **MUST** satisfy the rules for SOAP binding defined in section
455 “SOAP Binding”.
 - 456 2. The outgoing message **MUST** satisfy the rules for WS-Addressing SOAP binding
457 given in [WSAv1.0-SOAP].
 - 458 3. The outgoing message **MUST** include the mandatory header blocks defined above.
 - 459 4. All other Liberty headers defined in [LIB-SOAP] **SHOULD NOT** be used with this
460 profile since implementations of the profile are not required to support them.
 - 461 5. Each header block included in the outgoing message **MUST** conform to the
462 processing rules defined for each header block.
- 463

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

- 469
- 470 1. Construct the XML payload to be included in the SOAP Body.
 - 471 2. Construct a SOAP envelope with `<Header>` and `<Body>`, and embed the payload in
472 the `<Body>`. Add a `wsu:Id` attribute⁴ to the `<Body>` element.
 - 473 3. Add a `<wsa:MessageID>` header block (including a `wsu:Id` attribute) which
474 uniquely identifies the message; for example generate a 160-bit pseudorandom
475 number and embed it in a URI as follows:
476
477 `http://spwsp.com/ffeeddccbbaa99887766 554433221100ffeebbcc`
478
 - 479 4. When generating a response, include a `<wsa:RelatesTo>` element (including a
480 `wsu:Id` attribute) containing the message ID of the request.
 - 481 5. Add a `<wsa:Action>` header block (including a `wsu:Id` attribute) corresponding to
482 the `SOAPAction` HTTP header as required by the service being invoked.

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

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

520 Below is shown an example SOAP message that is compliant with the Liberty Basic SOAP
521 binding:
522

```
523 <?xml version="1.0" encoding="UTF-8"?>
524 <s:Envelope
525   xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
526   xmlns:sbf="urn:liberty:sb"
527   xmlns:sbfprofile="urn:liberty:sb:profile"
528   xmlns:sec="urn:liberty:security:2006-08"
529   xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
530     1.0.xsd"
531   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
532     1.0.xsd"
533   xmlns:wsa="http://www.w3.org/2005/08/addressing"
534   xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
535
536   <s:Header>
537     <wsa:MessageID wsu:Id="mid">f63d289c-cd9a-4c00-bf87-c4bad0310646</wsa:MessageID>
538
539     <wsa:To wsu:Id="to">...</wsa:To>
540
541     <wsa:Action wsu:Id="action">urn:liberty:id-sis-pp:2003-08:Modify</wsa:Action>
542
543
544     <sbfc:Framework
545       version="2.0"
546       sbfcprofile:profile="urn:liberty:sb:profile:basic"
547       s:mustUnderstand="1"
548       s:actor="http://schemas.../next"
549       wsu:Id="framework"/>
550
551
552     <wsse:Security mustUnderstand="1">
553       <wsu:Timestamp wsu:Id="ts">
554         <wsu:Created>2008-08-17T04:49:17Z</wsu:Created >
555       </wsu:Timestamp>
556
557       <!-- this is the holder-of-key token with the sender's certificate -->
558       <saml2:Assertion
559         xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
560         Version="2.0"
561         ID="sxJu9g/vvLG9sAN9bKp/8q0NKU="
562         IssueInstant="2008-08-01T16:58:33Z">
563         <saml2:Issuer>http://authority.example.com/</Saml2:Issuer>
564
565         <!-- signature by the issuer over the assertion -->
566         <ds:Signature>
567           <ds:SignedInfo>
568             <ds:CanonicalizationMethod
569               Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
570             <ds:SignatureMethod
571               Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
572             <ds:Reference URI="#sxJu9g/vvLG9sAN9bKp/8q0NKU=">
573               <ds:Transforms>
574                 <ds:Transform
575                   Algorithm="http://www.w3.org/2000/09/xmldsig#envelopedsignature" />
576                 </ds:Transforms>
```



```
577         <ds:DigestMethod
578             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
579
580         <ds:DigestValue>TCDVSuG6grhyHbzhQFWFzGrxIPE=</ds:DigestValue>
581     </ds:Reference>
582 </ds:SignedInfo>
583 <ds:SignatureValue>
584     x/GyPbzmFEe85pGD3claXG4Vspb9V9jGCjwcRCKrtwPS6vdVNCcY5rHaFPYWkf+5
585     EIYcPzx+pXlh43SmwviCqXRjRtMANWbHLhWAptaKlywS7gFgsD01qjyen3CP+m3D
586     w6vKhaqledl0BYyrIzb4KkHO4ahNyBVXbJwqv5pUaE4=
587 </ds:SignatureValue>
588 <ds:KeyInfo>
589     <ds:X509Data>
590 <!-- data identifying the signer's certificate -->
591     </ds:X509Data>
592 </ds:KeyInfo>
593 </ds:Signature>
594
595
596 <saml2:Subject>
597     <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">
598         005a06e0-ad82-110d-a556-004005b13a2b
599     </saml2:NameID>
600
601     <!-- Here comes the subject confirmation method saying this is a holder-of-
602 key -->
603
604     <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-
605 key">
606
607         <!-- Here comes a NameID indicating the ID of the sender who must confirm
608 with a key -->
609
610         <saml2:NameID format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
611             http://wsc.someorg.com
612         </saml2:NameID>
613
614         <!-- Here comes info on the key to confirm with (same as signing key) -->
615
616         <saml2:SubjectConfirmationData
617             xsi:type="saml2:KeyInfoConfirmationDataType"
618             <ds:KeyInfo>
619                 <ds:X509Data>
620                     <!-- Here comes the sender's X509 cert -->
621                     MIIB9zCCAWSgAwIBAgIQ...
622                 </ds:X509Data>
623             </ds:KeyInfo>
624         </saml2:SubjectConfirmationData>
625     </saml2:SubjectConfirmation>
626 </saml2:Subject>
627
628     <!-- Entity which should consume the information in the assertion. -->
629 <saml2:Conditions
630     NotOnOrAfter="2008-08-01T21:42:43Z">
631     <saml2:AudienceRestrictionCondition>
632     <saml2:Audience>http://wsp.example.com</saml2:Audience>
```

```
631         </saml2:AudienceRestrictionCondition>
632     </saml2:Conditions>
633
634     <saml2:AttributeStatement>
635         ...
636     </saml2:AttributeStatement>
637 </saml2:Assertion>
638
639     <!-- This SecurityTokenReference is used to reference the SAML Assertion from a
640 ds:Reference -->
641     <wsse:SecurityTokenReference
642         xmlns:wsse="..." xmlns:wsu="..." xmlns:wssell="..."
643         wsu:Id="str1"
644         wssell:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
645 1.1#SAMLV2.0">
646         <!-- A key identifier with the SAML Assertion ID -->
647         <wsse:KeyIdentifier
648             ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
649 1.1#SAMLID">
650             sxJu9g/vvLG9sAN9bKp/8q0NKU=
651         </wsse:KeyIdentifier>
652     </wsse:SecurityTokenReference>
653
654
655     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
656         <ds:SignedInfo>
657             <!-- in general include a ds:Reference for each wsa: header added according
658 to SOAP binding -->
659
660             <!-- include the MessageID in the signature -->
661             <ds:Reference URI="#mid">...</ds:Reference>
662
663             <!-- include the To in the signature -->
664             <ds:Reference URI="#to">...</ds:Reference>
665
666             <!-- include the Action in the signature -->
667             <ds:Reference URI="#action">...</ds:Reference>
668
669             <!-- include the Framework in the signature -->
670             <ds:Reference URI="#framework">...</ds:Reference>
671
672             <!-- include the Timestamp in the signature -->
673             <ds:Reference URI="#ts">...</ds:Reference>
674
675             <!-- include the SAML Assertion in the signature to avoid token substitution
676 attacks -->
677             <ds:Reference URI="#str1">
678                 <ds:Transform Algorithm="http://docs.oasis-open.org/wss/2004/01/oasis-
679 200401-wsssoap-message-security-1.0#STR-Transform">
680                     <wsse:TransformationParameters>
681                         <ds:CanonicalizationMethod
682                             Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
683                     </wsse:TransformationParameters>
684                 </ds:Transform>
```

```

685         </ds:Reference>
686
687         <!-- bind the body of the message -->
688         <ds:Reference URI="#MsgBody">
689             <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
690             <ds:DigestValue>YgGfS0pi56pu...</ds:DigestValue>
691         </ds:Reference>
692     </ds:SignedInfo>
693
694     <!-- include a security token reference for holder-of-key confirmation -->
695     <ds:KeyInfo>
696         <wsse:SecurityTokenReference
697             xmlns:wsse="..." xmlns:wsu="..." xmlns:wssell="..."
698             wsu:Id="str2"
699             wssell:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-
700 profile-1.1#SAMLV2.0">
701         <!-- A key identifier with the SAML Assertion ID -->
702         <wsse:KeyIdentifier
703             ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
704 1.1#SAMLID">
705             sxJu9g/vvLG9sAN9bKp/8q0NKU=
706         </wsse:KeyIdentifier>
707     </wsse:SecurityTokenReference>
708 </ds:KeyInfo>
709
710     <ds:SignatureValue>
711         HJJWbvqW9E84vJVQkjJLLA6nNvBX7mY00TZhWbDFNDElgsCSXZ5Ekw==
712     </ds:SignatureValue>
713 </ds:Signature>
714 </wsse:Security>
715 </s:Header>
716
717 <s:Body wsu:Id="MsgBody">
718     <idpp:Modify>
719         : <!-- this is an ID-PP Modify message -->
720     </idpp:Modify> </s:Body>
721 </s:Envelope>

```

722 4.2 Receiving and processing a SOAP message

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

725

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

728

- 729 1. The incoming message MUST satisfy the rules for SOAP binding defined in section
730 "SOAP Binding".
- 731 2. The incoming message MUST satisfy the rules given in [WSAv1.0-SOAP].
- 732 3. The incoming message MUST include all mandatory header blocks defined above.
- 733 4. Each header block in the message (mandatory as well as optional) MUST be tested
734 according to the processing rules defined above.

735

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

742

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

777

778 Note that the recipient may need to perform additional checks e.g. related to authorization.
779

780 **5 Security Considerations**

781 Message integrity and authenticity is established by mandatory signing (and subsequent
782 verification) of the SOAP body, header blocks in this specification and security tokens.
783

784 Message confidentiality is not addressed directly in this profile but may be established by
785 using a secure transport protocol such as SSL 3.0, TLS 1.1 or later HTTPS, or by
786 encryption of name identifiers or individual attributes in the SAML 2.0 assertion.
787

788 Message freshness and prevention against replay attacks is established by including unique
789 message Ids that WSP's should cache, time stamps and expiry of tokens. How long time a
790 message should be kept in the cache at the WSP is governed by deployment policy.
791

792 Message authorization is established by including signed authentication assertions in the
793 form of SAML assertions issued by a trusted STS, Liberty Discovery Service or Identity
794 Provider.
795

796 Security tokens in the form of SAML 2.0 assertions are signed by the issuer and sensitive
797 attributes may be encrypted if deemed necessary via the mechanisms described in [SAML-
798 CORE] including encryption of the entire assertion, name identifiers and individual
799 attributes.
800

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

- 804 • The sender identity as established via the signature.
- 805 • The invoker / user identity as established via authentication assertions.
- 806 • The resource / service being accessed.
- 807 • Trust in the STS, Discovery Service or Identity Provider that has issued the
808 authentication assertion.
- 809 • The assurance level established as part of the assertion.

810 **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).

811