



SAMLv2.0: HTTP POST “SimpleSign” Binding

~~Committee-Draft 021, 105 October~~ January 2007~~6~~

Document identifier:

draft-sstc-saml-binding-simplesign-~~ed-021~~

Location:

http://www.oasis-open.org/committees/documents.php?wg_abbrev=security

Technical Committee:

OASIS Security Services TC

Chairs:

Hal Lockhart, BEA Systems, Inc.
Prateek Mishra, Oracle Corporation

Editors:

Jeff Hodges, NeuStar
Scott Cantor, Internet2

Abstract:

This specification defines a SAML HTTP protocol binding, specifically using the HTTP POST method, and not using XML Digital Signature for SAML message data origination authentication. Rather, a “sign the BLOB” technique is employed wherein a conveyed SAML message is treated as a simple octet string if it is signed. Conveyed SAML assertions may be individually signed using XMLdsig. Security is optional in this binding.

Status:

This is a **Committee Working Draft** ~~approved by the Security Services Technical Committee on 10 October 2006 and the text may change before completion.~~

Committee members should submit comments and potential errata to the security-services@lists.oasis-open.org list. Others should submit them by filling out the web form located at http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=security.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights web page for the Security Services TC (<http://www.oasis-open.org/committees/security/ipr.php>).

35 **Table of Contents**

36 1 Introduction..... 3

37 1.1 Protocol Binding Concepts..... 3

38 1.2 Notation..... 3

39 2 HTTP POST Binding-SimpleSign..... 5

40 2.1 Required Information..... 5

41 2.2 Overview..... 5

42 2.3 Relay State..... 5

43 2.4 Message Encoding and Conveyance..... 6

44 2.5 SimpleSign Signature..... 7

45 2.6 SimpleSign Signature Verification..... 7

46 2.7 Message Exchange..... 8

47 2.7.1 HTTP and Caching Considerations..... 10

48 2.7.2 Security Considerations..... 10

49 2.8 Error Reporting..... 10

50 2.9 Metadata Considerations..... 11

51 2.10 Note to Implementors..... 11

52 2.11 Example..... 11

53 3 References..... 14

54 Appendix A. Acknowledgments..... 15

55 Appendix B. Notices..... 16

1 Introduction

56

57 This specification defines a SAML HTTP protocol binding, specifically using the HTTP POST method, and
58 which specifically does not use XML Digital Signature[XMLSig] for SAML message data origination
59 authentication. Rather, a “sign the BLOB” technique is employed wherein a conveyed SAML message,
60 along with any content (e.g. SAML assertion(s)), is treated as a simple octet string if it is signed.
61 Additionally, it is out of the scope of this specification whether or not conveyed SAML assertions are
62 authenticated via XML Digital Signature. Security is optional in this binding.

63 The next subsection gives a general overview of SAML Protocol Binding concepts, followed by notation
64 and namespace declarations. The binding itself is defined in Section 2.

1.1 Protocol Binding Concepts

65

66 Mappings of SAML request-response message exchanges onto standard messaging or communication
67 protocols are called SAML *protocol bindings* (or just *bindings*). An instance of mapping SAML request-
68 response message exchanges into a specific communication protocol <FOO> is termed a <FOO> *binding*
69 *for SAML* or a *SAML <FOO> binding*.

70 For example, a SAML SOAP binding describes how SAML request and response message exchanges
71 are mapped into SOAP message exchanges.

72 The intent of this specification is to specify the given binding in sufficient detail to ensure that
73 independently implemented SAML-conforming software can interoperate when using standard messaging
74 or communication protocols.

75 Unless otherwise specified, this binding should be understood to support the transmission of any SAML
76 protocol message derived from the **samlp:RequestAbstractType** and **samlp:StatusResponseType**
77 types. Further, when this binding refers to "SAML requests and responses", it should be understood to
78 mean any protocol messages derived from those types.

79 For other terms and concepts that are specific to SAML, refer to the SAML glossary [SAMLGloss].

1.2 Notation

80

81 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD
82 NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as
83 described in IETF RFC 2119 [RFC2119].

84 `Listings of productions or other normative code appear like this.`

85

86 `Example code listings appear like this.`

87 **Note:** Notes like this are sometimes used to highlight non-normative commentary.

88 Conventional XML namespace prefixes are used throughout this specification to stand for their respective
89 namespaces as follows, whether or not a namespace declaration is present in the example:

Prefix	XML Namespace	Comments
saml:	urn:oasis:names:tc:SAML:2.0:assertion	This is the SAML V2.0 assertion namespace [SAMLCore].
samlp:	urn:oasis:names:tc:SAML:2.0:protocol	This is the SAML V2.0 protocol namespace [SAMLCore].

Prefix	XML Namespace	Comments
SOAP-ENV:	http://schemas.xmlsoap.org/soap/envelope	This namespace is defined in SOAP V1.1 [SOAP11].

90

91 This specification uses the following typographical conventions in text: `<ns:Element>`, `XMLAttribute`,
 92 **Datatype**, `OtherKeyword`. In some cases, angle brackets are used to indicate non-terminals, rather than
 93 XML elements; the intent will be clear from the context.

2 HTTP POST Binding-SimpleSign

94

95 The HTTP POST binding, defined in [SAMLBind], defines a mechanism by which SAML protocol
96 messages may be transmitted within the base64-encoded content of an HTML form control. When using
97 that binding, SAML protocol messages and/or SAML assertions are signed using [XMLSig], which is an
98 XML-aware, XML-based, invasive digital signature paradigm necessitating canonicalization of the
99 signature target.

100 This document specifies an alternative HTTP POST-based binding where the conveyed SAML protocol
101 messages – including their content, i.e. any conveyed SAML assertions – are signed as simple “BLOBs”
102 (“Binary Large Objects”, aka binary octet strings).

103 Note that this binding defines the conveyance of an individual SAML request or response message via
104 HTTP POST. Thus this binding MAY be composed with the HTTP Redirect binding (see Section 3.4 of
105 [SAMLBind]) or the HTTP Artifact binding (see Section 3.6 of [SAMLBind]) to transmit request and
106 response messages in an overall SAML protocol exchange, the definition of which is termed a “SAML
107 Profile” [SAMLProf], using two different bindings.

2.1 Required Information

108

1.2.1 Required Information

109

110 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST-SimpleSign

111 **Contact information:** security-services-comment@lists.oasis-open.org

112 **Description:** Given below.

113 **Updates:** None. Rather, it provides an alternative to the HTTP POST Binding defined in [SAMLBind]

2.2 Overview

114

1.2.2 Overview

115

116 The HTTP POST-SimpleSign binding is intended for cases in which the SAML requester or responder
117 need to communicate using an HTTP user agent (as defined in HTTP 1.1 [RFC2616] as an intermediary,
118 and when data origination authentication and integrity protection of the SAML message is not required, or
119 when a lighter-weight signature mechanism (as compared to [XMLSig]) is appropriate. This may be
120 necessary, for example, if the communicating parties do not share a direct path of communication. It may
121 also be needed if the responder requires an interaction with the user agent in order to fulfill the request,
122 such as when the user agent must authenticate to it.

123 Note that some HTTP user agents may have the capacity to play a more active role in the protocol
124 exchange and may support other bindings that use HTTP, such as the SOAP and Reverse SOAP
125 bindings. This binding does not require such capabilities—it assumes nothing apart from the capabilities
126 of a common web browser.

127 | 2.3 Relay State

128 | 1.2.3 RelayState

129 RelayState data MAY be included with a SAML protocol message transmitted with this binding. The value
130 MUST NOT exceed 80 bytes in length and ~~SHOULD be integrity protected by the entity creating the~~
131 ~~message, either via a digital signature (see section 2.5) or by some independent means. SHOULD be~~
132 ~~integrity protected by the entity creating the SAML protocol message independent of any other protections~~
133 ~~that may or may not exist during message transmission. Signing is not realistic given the space limitation,~~
134 ~~but because the value is exposed to third-party tampering, the entity SHOULD ensure that the value has~~
135 ~~not been tampered with by using a checksum, a pseudo-random value, or similar means.~~

136 If a SAML request message is accompanied by RelayState data, then the SAML responder MUST return
137 its SAML protocol response message using a binding that also supports a RelayState mechanism, and it
138 MUST place the exact data it received with the request into the corresponding RelayState parameter in
139 the response message.

140 If no such value is included with a SAML request message, or if the SAML response message is being
141 generated without a corresponding request, then the SAML responder MAY include RelayState data to be
142 interpreted by the recipient based on the use of a profile or prior agreement between the parties.

143 | 2.4

144 | 2.4.1 Message Encoding and Conveyance

145 This section describes how to encode a SAML protocol message, and thus any SAML assertion(s) it may
146 contain, into HTML FORM "control(s)" [HTML401] (Section 17), thus enabling the SAML protocol
147 message to be conveyed via the HTTP POST method.

148 A SAML protocol message is form-encoded by:

- 149 1. Applying the base-64 encoding rules to the XML representation of the message. The resulting
150 base64-encoded value MAY be line-wrapped at a reasonable length in accordance with common
151 practice.
- 152 2. Encoding the result from the prior step into a "form data set", in the same fashion as is specified for
153 "successful controls" in [HTML401] (Section 17.13.3), as a form "control value". The HTML
154 document also MUST adhere to the XHTML specification, [XHTML].
 - 155 a. If the SAML protocol message is a SAML request, then the form "control name" used to convey
156 the SAML protocol message itself MUST be `SAMLRequest`.
 - 157 b. If the SAML protocol message is a SAML response, then the form "control name" used to
158 convey the SAML protocol message itself MUST be `SAMLResponse`.
 - 159 c. Any additional form controls or presentation, other than those noted below for including a
160 signature, MAY be included but MUST NOT be required in order for the recipient to nominally
161 process the SAML protocol message itself.

162 SAML protocol messages, and any SAML assertions contained within the SAML protocol messages,
163 MAY be signed using [XMLSig], and if so, any such signatures MUST remain intact. Additionally, SAML
164 protocol messages MAY be signed using the technique given below in section 2.0.5. This technique is
165 referred to as the "SimpleSign technique". The SimpleSign signature value is conveyed in a form control
166 value named `Signature`, and the signature algorithm is conveyed in a form control value named
167 `SigAlg`. These form control values are included in the form data set constructed in step 2 above.

168 If the SAML protocol message is signed using SimpleSign, the `Destination` XML attribute in the root
169 SAML element of the SAML protocol message MUST contain the URL to which the sender has instructed

170 the user agent to deliver the message. The recipient MUST then verify that the value matches the location
171 at which the SAML protocol message has been received. Also, the signer's certificate or other keying
172 information MAY be included in a form control named `KeyInfo`. This form control, if present, MUST
173 contain a base-64 encoded `<ds:KeyInfo>` element (~~[XMLSig]~~ (base-64 encoding is done as in step 1,
174 above). ~~This form control MUST NOT be included in the signed form data set constructed in step 2 above.~~

175 If a "RelayState" value is to accompany the SAML protocol message, it MUST be in a form control named
176 `RelayState`, and included in the form data set constructed in step 2 above, and also included in any
177 signed content if the message is signed.

178 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using
179 this binding to which the SAML protocol message is to be delivered. The `method` attribute MUST be
180 "POST". The `enctype` attribute specifies the form content type and MUST be `application/x-www-`
181 `form-urlencoded`.

182 All of the above form attributes and form controls, to which values are assigned per the above discussion,
183 comprise the form data set. The form data set is then encoded into an HTTP response `message-body`
184 as a `<FORM>` element. The HTTP response message is then sent to the user agent.

185 Any technique supported by the user agent MAY be used to cause the submission of the form (to cause it
186 to be conveyed to the SAML protocol message recipient), and any form content necessary to support this
187 MAY be included, such as submit controls and client-side scripting commands. However, the recipient
188 MUST be able to process the message without regard for the mechanism by which the form submission is
189 initiated.

190 Note that any form control values included MUST be transformed so as to be safe to include in the
191 XHTML document. This includes transforming characters such as quotes into HTML entities, etc.
192 [HTML401][XHTML]

193 2.5

194 2.5.1 SimpleSign Signature

195 To construct a signature of a SAML message conveyed by this binding:

- 196 1. The signature algorithm used MUST be identified by a URI, specified according to [XMLSig] or
197 whatever specification governs the algorithm. The following signature algorithms (see [XMLSig])
198 and their URI representations MUST be supported with this encoding mechanism:
 - 199 • DSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#dsa-sha1>
 - 200 • RSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#rsa-sha1>
- 202 2. ~~A string consisting of the concatenation of the raw, unencoded XML making up the SAML protocol~~
203 ~~message (NOT the base64-encoded version), the RelayState value (if present), and the~~
204 ~~SigAlg value, is constructed~~~~4 above, is constructed0., and SAMLRequest (or~~
205 ~~SAMLResponse) values (as appropriate), as defined in section 2. SigAlg~~
206 ~~(if present), RelayState~~~~A string consisting of the concatenation of the~~ in one of the
207 following ways (each individually ordered as shown):

```
208 SAMLRequest=value&RelayState=value&SigAlg=value  
209  
210 SAMLResponse=value&RelayState=value&SigAlg=value  
211
```

213 3.

214 4. The resultant octet string is fed into the signature algorithm.

215 5. The value yielded by the signature algorithm is base64 encoded (see [RFC2045]), and used as the
216 value for the `Signature` form control as discussed in section 2.0.4, above.

217 -

218 **2.5.2 Note that this is subtly different from the signature approach defined**
219 **by the HTTP-Redirect binding [SAMLBind]. Experimentation shows**
220 **that many web browsers alter linefeeds when submitting form**
221 **controls that span multiple lines. Since base64-encoded data often**
222 **wraps, it is not possible to guarantee that the values submitted will**
223 **match what the original signer produced, resulting in verification**
224 **failures. Using the raw XML content as a component of the octet**
225 **string addresses this issue.**

226 The original XML MUST be concatenated with the other information as shown above without regard for
227 any embedded whitespace, even if the result spans multiple lines. The specific whitespace characters
228 present will be safely encoded in base64 and then recovered by the relying party for use in verifying the
229 signature.

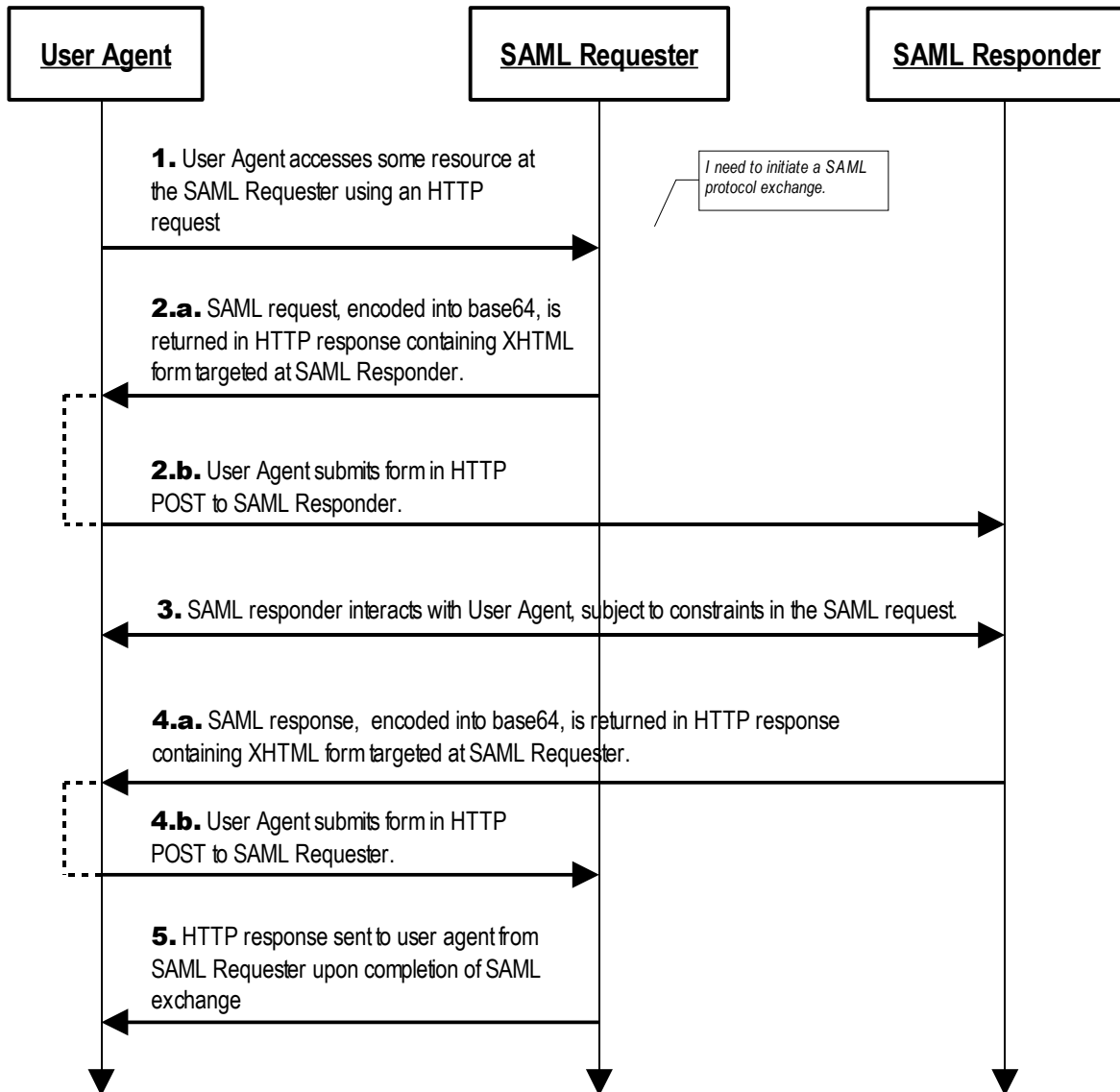
230 2.6 SimpleSign Signature Verification

231 To verify a received SAML protocol message, which was signed using SimpleSign and conveyed by this
232 binding, the receiver MUST extract the form control values for the `RelayState` (if present), `SigAlg`,
233 `KeyInfo` (if present), and `SAMLRequest` (or `SAMLResponse`) values (as appropriate) from the received
234 HTTP message. Then the receiver reconstructs the string as described in section 2.0.5 step 2, above.
235 The signature value conveyed in the `Signature` control value is then checked against this string per the
236 signature algorithm given by the `SigAlg` control value, and using (as appropriate, see [XMLSig]) the
237 keying material obtained via the `<ds:KeyInfo>` conveyed in the `KeyInfo` control value (if present).
238 Error handling and generated messages as a result of the signature not verifying are implementation-
239 dependent.

240 2.7 ~~Message Exchange~~

241 2.7.1 ~~Message Exchange~~

242 The system model used for SAML conversations via this binding is a request-response model. However,
243 a SAML request message is sent to the user agent via an HTTP response message, and subsequently
244 delivered to the SAML responder via an HTTP request message issued by the user agent. Any HTTP
245 interactions before, between, and after the foregoing ~~exchanges~~ take place is unspecified. Both the
246 SAML requester and responder are assumed to be HTTP responders. See the following diagram
247 illustrating the messages exchanged. Note that although the diagram illustrates both the SAML request
248 and the SAML response being conveyed via the HTTP POST-SimpleSign binding, one or the other of the
249 SAML request or the SAML response could be conveyed via a different SAML HTTP-based binding.



- 251 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
252 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 253 2. (a) The system entity acting as a SAML requester responds to an HTTP request from the user
254 agent by returning a SAML request. The request is returned in an XHTML document containing the
255 form and content defined in Section 2.0-4, above. (b) The user agent delivers the SAML request by
256 issuing an HTTP POST request to the SAML responder.
- 257 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
258 SAML response or it MAY return arbitrary content to facilitate subsequent interaction with the user
259 agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to
260 indicate the requester's level of willingness to permit this kind of interaction (for example, the
261 `IsPassive` attribute in `<samlp:AuthnRequest>` [SAMLCore]).
- 262 4. Eventually the responder SHOULD (a) return a SAML response to the user agent to be (b) returned
263 to the SAML requester. The SAML response is returned in the same fashion as described for the
264 SAML request in step 2, if this or a similar binding is employed for this step. Otherwise, details may
265 vary.

266 5. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
267 user agent.

268 1.

269 2.7.2 HTTP and Caching Considerations

270 HTTP proxies and the user agent intermediary should not cache SAML protocol messages. To ensure
271 this, the following rules SHOULD be followed.

272 When returning SAML protocol messages using HTTP 1.1, HTTP responders SHOULD:

- 273 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 274 • Include a `Pragma` header field set to "no-cache".

275 There are no other restrictions on the use of HTTP headers.

276 1.2.3.1

277 2.7.3 Security Considerations

278 The presence of the user agent intermediary means that the requester and responder cannot rely on the
279 transport layer for endpoint-to-endpoint (i.e. SAML Requester to/from SAML Responder) authentication,
280 integrity or confidentiality protection. This binding defines [the SimpleSign approach](#) as a means for
281 signing the conveyed SAML protocol messages and optional `RelayState` in order to provide endpoint-
282 to-endpoint integrity protection and data origin authentication.-

283 This binding SHOULD NOT be used if the content of the request or response should not be exposed to
284 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is
285 OPTIONAL and depends on the environment of use. If on-the-wire confidentiality is necessary, SSL 3.0
286 [SSL3] or TLS 1.0 [RFC2246] SHOULD be used to protect the overall HTTP messages, and the conveyed
287 SAML protocol messages, in transit between the user agent and the SAML requester and responder.

288 In general, this binding relies on message-level authentication and integrity protection via signing and
289 does not support confidentiality of messages from the user agent intermediary.

290 **NOTE:** eCryptographically-based security is entirely OPTIONAL in this binding. If no
291 security mechanisms are employed, then there is essentially no runtime assurance as to
292 the identity of any of the communicating entities.

293 If the SAML protocol messages are signed (using [the SimpleSign approach](#) or [XMLSig]) then the
294 `Destination` XML attribute in the root SAML element of the SAML protocol message MUST contain the
295 URL to which the sender has instructed the user agent to deliver the message. The recipient MUST then
296 verify that the value matches the location at which the message has been received.

297 Note also that the SimpleSign technique, if employed, binds the `RelayState` value (if present) to the SAML
298 protocol message, unlike the [XMLSig]-based technique of the HTTP POST binding [SAMLBind]. Thus, if
299 a SAML protocol message is not signed using SimpleSign, but is signed using the [XMLSig]-based
300 technique, then the caveats with respect to any conveyed `RelayState` value, presented in section 3.5.5.2
301 of [SAMLBind], should be taken into account.

302 2.7.4

303 2.8 Error Reporting

304 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD

305 return a response message with a second-level `<samlp:StatusCode>` value of
306 `urn:oasis:names:tc:SAML:2.0:status:RequestDenied`.

307 HTTP interactions during the message exchange MUST NOT use HTTP error status codes to indicate
308 failures in SAML processing, since the user agent is not a full party to the SAML protocol exchange.

309 For more information about SAML status codes, see the SAML assertions and protocols specification
310 [SAMLCore]

311 | 1.2.4

312 | 2.9 Metadata Considerations

313 Support for the HTTP POST-SimpleSign binding SHOULD be reflected by indicating URL endpoints at
314 which requests and responses for a particular protocol or profile should be sent. Either a single endpoint
315 or distinct request and response endpoints MAY be supplied [SAMLMeta]. The identification URI given in
316 section 2.0.1 is used as the value for the `Binding` attribute of any **endpoint** elements.

317 | 2.9.1

318 | 2.10 Note to Implementors

319 SAML protocol message recipients can distinguish between HTTP-SAML messages constructed via this
320 specification's HTTP POST-SimpleSign binding and ones constructed via the HTTP POST binding
321 [SAMLBind] by examining received HTTP messages for an XHTML form field with a name attribute value
322 of `signature`. If this is present, then the message MUST be processed in accordance with this
323 specification. If not present, then the HTTP message MAY be processed in accordance with the HTTP
324 POST binding specification.

325 | -

326 | 2.11 Example ~~Using HTTP POST-SimpleSign SAML Message Exchange~~

327 In this example, a `<LogoutRequest>` and `<LogoutResponse>` message pair is exchanged using the
328 HTTP POST-SimpleSign binding. The messages are signed as described in section 2.0.5, above. If the
329 messages were unsigned, they would be the same as shown below, except that the hidden form controls
330 named `Signature` and `SigAlg` would be missing.

331 First, here are the actual SAML protocol messages being exchanged:

```
332 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
333 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
334 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
335 21T19:00:49Z" Version="2.0">  
336 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
337 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
338 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
339 <samlp:SessionIndex>1</samlp:SessionIndex>  
340 </samlp:LogoutRequest>
```

341

```
342 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
343 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
344 ID="b0730d21b628110d8b7e004005b13a2b"  
345 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"  
346 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">  
347 <Issuer>https://ServiceProvider.com/SAML</Issuer>  
348 <samlp:Status>
```

```

349     <samlp:StatusCode
350 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
351     </samlp:Status>
352 </samlp:LogoutResponse>
353

```

354 The initial HTTP request from the user agent in step 1 is not defined by this binding. To initiate the logout
355 protocol exchange, the SAML requester returns the following HTTP response, containing a SAML request
356 message. The `SAMLRequest` parameter value is actually derived from the request message above.

```

357 HTTP/1.1 200 OK
358 Date: 21 Jan 2004 07:00:49 GMT
359 Content-Type: text/html; charset=iso-8859-1
360
361 <?xml version="1.0" encoding="UTF-8"?>
362 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
363 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
364 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
365 <body onload="document.forms[0].submit()">
366
367 <noscript>
368 <p>
369 <strong>Note:</strong> Since your browser does not support JavaScript,
370 you must press the Continue button once to proceed.
371 </p>
372 </noscript>
373
374 <form action="http://ServiceProvider.com/SAML/SLO/Browser" method="post">
375 <div>
376 <input type="hidden" name="RelayState"
377 value="0043bfc1bc45110dae17004005b13a2b"/>
378 <input type="hidden" name="SAMLRequest"
379 value="PHNhbWxwOkxvZ291dFJlcXVlc3QgeG1sbnM6c2FtbHA9InVybjpvczpuYW11
380 czp0YzptQU1MOjIuMDpwcm90b2NvbCIgeG1sbnM9InVybjpvczpuYW11czp0
381 YzptQU1MOjIuMDphc3NlcnRpb24iCiAgICBjRD0iZDZiN2MzODhjZWZmNmZhN2Mz
382 OWMyOGZkMjk4NjQ0YTgiIElzc3VlSW5zdGFudD0iMjAwNC0wMS0yMVQxOTowMDo0
383 OVoiIFZlcnNpb249IjIuMCI+CiAgICAgICA8SXNzdWVyPmh0dHBzOi8vSWRlbnRpdHlQ
384 cm92aWRlci5jb20vU0FNTDdwSXNzdWVyPggogICAgPE5hbWVJRCBGB3JtYXQ9InVy
385 bjpvYXNpczpuYW11czp0YzptQU1MOjIuMDpuYW11aWQzM9ybWF0OnBlcnNpc3Rl
386 bnQiPjAwNWUwLWwLWFkODI0MTEwZC1hNTU2LTAwNDAwNW1xM2EyYjwvTmFtZU1E
387 PggogICAgPHNhbWxwO1Nlc3Npb25JbmlleD4xPC9zYW1scDpTZXRzaW9uSW5kZXg+
388 Cjwvc2FtbHA6TG9nb3V0UmVxdWVzdD4K"/>
389 <input type="hidden" name="Signature"
390 value="J4if7CCeHVfn4H6hmZN5fijOjQIyZ/1aoFUZWz4LCRN3J82UeoyYvAiTD0QOUZHT
391 RJNU1lWGublpW4QR9MH5bwfLEa8XDivA118dR0Q7YN5L/U5rmbxnglQ9pV0jt44c
392 RNeqtbLW0YF4plfcqg7E5iOSljE3QLkiaAdkAec2a4HwPFkn/JP7w011Mc6kU8ML
393 CBbZAa3+94ZvVwHBEdyCdU+lyEvf+JGxTw66BwI2ugmPfxvoJds00AWwS3KhAFhL
394 LSPXnhb3nd/ovKNNV/khZYwqsFTFNTMA+0JraKsZiCrTEZzEPXaP9KilrjPIIvRV
395 xDQhETj96flk5zMkEM3ruw==" />
396 <input type="hidden" name="SigAlg"
397 value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
398 </div>
399 <noscript>
400 <div>
401 <input type="submit" value="Continue"/>
402 </div>
403 </noscript>
404 </form>
405 </body>
406 </html>
407

```

408 After any unspecified interactions may have taken place, the SAML responder returns the HTTP response
409 below containing the SAML response message. Again, the `SAMLResponse` parameter value is actually
410 derived from the response message above.

```
411 HTTP/1.1 200 OK
412 Date: 21 Jan 2004 07:00:49 GMT
413 Content-Type: text/html; charset=iso-8859-1
414
415 <?xml version="1.0" encoding="UTF-8"?>
416 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
417 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
418 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
419 <body onload="document.forms[0].submit()">
420
421 <noscript>
422 <p>
423 <strong>Note:</strong> Since your browser does not support JavaScript,
424 you must press the Continue button once to proceed.
425 </p>
426 </noscript>
427
428 <form action="https://IdentityProvider.com/SAML/SLO/Response"
429 method="post">
430 <div>
431 <input type="hidden" name="RelayState"
432 value="0043bfc1bc45110dae17004005b13a2b"/>
433 <input type="hidden" name="SAMLResponse"
434 value="PHNhbWxwOkxvZ291dFJlcXVlc3QgeG1sbnM6c2FtbHA9InVybJpYXNpczpuYW1l
435 czp0YzptQU1MOjIuMDpwc3NlcnRpb24iCiAgICBjRD0iZDZiN2MzODhjZWZmNmZhN2Mz
436 YzptQU1MOjIuMDpwc3NlcnRpb24iCiAgICBjRD0iZDZiN2MzODhjZWZmNmZhN2Mz
437 OWMYOGZkMjk4NjQ0YTgiIElzc3VlSW5zdGFudD0iImJAwNC0wMS0yMVQxOTowMDo0
438 OVoiIFZlcnNpb249IjIuMCI+CiAgICAgICA8SXMzZdWVYpPm0dHBzOi8vSWRlbnRpdHlQ
439 cm92aWRlci5jb20vU0FNTDdwSXNzdWVYpPogICAgPE5hbWVJRCBGb3JtYXQ9InVy
440 bjpvYXNpczpuYW1lczp0YzptQU1MOjIuMDpYw1laWQtZm9ybWF0OnBlcnNpc3Rl
441 bnQiPjAwNWwWUwLWVfKODItMTEwZC1hNTU2LTAwNDAwNW1xM2EyYjwvTmFtZU1E
442 PggICAgPHNhbWxwOlNlc3Npb25JbWVleD4xPC9zYW1scDpTZXNzaW9uSW5kZXg+
443 Cjwvc2FtbHA6TG9nb3V0UmVxdWVzdD4K"/>
444 <input type="hidden" name="Signature"
445 value="DCDqAwIDqSwyXGvG2cYvNjmj7Plkt0+kbCfRjq9gGTrN4KKPxvQl5EsFrWRkMOdx
446 xuwPlDWPKvfGx6rt+pKwLgCt1TqRj+71y+VdGS8ORsBeEIURRn9wSu+pKsWiHexw
447 KnIe65bjONbg2db44QOWz1De76fLi05Psy/7HZTQuMoDRFYSR//VyNGHQmf9Sxi6
448 mkmrYMXPOyZAUfNhX4eLaXFfwCHt0yRrEcm/PAEDDa7uqe8Uo5ilstgXDWDodWdk
449 Szk8ZS1irjFkvtxH7FJlM9ADt1W/SoX92jGjMIrdQwCyArI6o8KtiDp/cjDjHZGi
450 XLx2WvS7GEibA7Qd+5hSBQ==" />
451 <input type="hidden" name="SigAlg"
452 value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
453 </div>
454 <noscript>
455 <div>
456 <input type="submit" value="Continue"/>
457 </div>
458 </noscript>
459 </form>
460 </body>
461 </html>
```

3 References

462

- 463 [HTML401] D. Raggett et al. *HTML 4.01 Specification*. World Wide Web Consortium
464 Recommendation, December 1999. See <http://www.w3.org/TR/html4>.
- 465 [RFC2045] N. Freed et al. *Multipurpose Internet Mail Extensions (MIME) Part One: Format*
466 *of Internet Message Bodies*, IETF RFC 2045, November 1996. See
467 <http://www.ietf.org/rfc/rfc2045.txt>.
- 468 [RFC2119] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF
469 RFC 2119, March 1997. See <http://www.ietf.org/rfc/rfc2119.txt>.
- 470 [RFC2246] T. Dierks et al. *The TLS Protocol Version 1.0*. IETF RFC 2246, January 1999.
471 See <http://www.ietf.org/rfc/rfc2246.txt>.
- 472 [RFC2616] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC 2616, June
473 1999. See <http://www.ietf.org/rfc/rfc2616.txt>.
- 474 [SAMLBind] S. Cantor et al. *Bindings for the OASIS Security Assertion Markup Language*
475 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-bindings-2.0-os.
476 See <http://www.oasis-open.org/committees/security/>.
- 477 [SAMLCore] S. Cantor et al. *Assertions and Protocols for the OASIS Security Assertion*
478 *Markup Language (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-
479 core-2.0-os. See <http://www.oasis-open.org/committees/security/>.
- 480 [SAMLGloss] J. Hodges et al. *Glossary for the OASIS Security Assertion Markup Language*
481 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-glossary-2.0-os.
482 See <http://www.oasis-open.org/committees/security/>.
- 483 [SAMLMeta] S. Cantor et al. *Metadata for the OASIS Security Assertion Markup Language*
484 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-metadata-2.0-os.
485 See <http://www.oasis-open.org/committees/security/>.
- 486 [SAMLProf] S. Cantor et al. *Profiles for the OASIS Security Assertion Markup Language*
487 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-profiles-2.0-os. See
488 <http://www.oasis-open.org/committees/security/>.
- 489 [SAMLSecure] F. Hirsch et al. *Security and Privacy Considerations for the OASIS Security*
490 *Assertion Markup Language (SAML) V2.0*. OASIS SSTC, March 2005.
491 Document ID saml-sec-consider-2.0-os. See [http://www.oasis-
open.org/committees/security/](http://www.oasis-
492 open.org/committees/security/).
- 493 [SOAP11] D. Box et al. *Simple Object Access Protocol (SOAP) 1.1*. World Wide Web
494 Consortium Note, May 2000. See [http://www.w3.org/TR/2000/NOTE-SOAP-
20000508/](http://www.w3.org/TR/2000/NOTE-SOAP-
495 20000508/).
- 496 [SSL3] A. Frier et al. *The SSL 3.0 Protocol*. Netscape Communications Corp, November
497 1996.
- 498 [SSTCWeb] OASIS Security Services Technical Committee website, [http://www.oasis-
open.org/committees/security](http://www.oasis-
499 open.org/committees/security).
- 500 [XHTML] *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*.
501 World Wide Web Consortium Recommendation, August 2002. See
502 <http://www.w3.org/TR/xhtml1/>.
- 503 [XMLSig] D. Eastlake et al. *XML-Signature Syntax and Processing*. World Wide Web
504 Consortium Recommendation, February 2002. See
505 <http://www.w3.org/TR/xmlsig-core/>.

506 Appendix A. Acknowledgments

507 The editors would like to acknowledge the contributions of the OASIS Security Services Technical
508 Committee, whose voting members at the time of publication were:

- 509
- 510 • Christopher Laskowski, Booz Allen Hamilton
 - 511 • Rebekah Metz, Booz Allen Hamilton
 - 512 • Hal Lockhart, BEA Systems, Inc.
 - 513 • Steve Anderson, BMC Software
 - 514 • Sharon Boeyen, Entrust
 - 515 • Thomas Wisniewski, Entrust
 - 516 • Carolina Canales-Valenzuela, Ericsson
 - 517 • Dana Kaufman, Forum Systems, Inc.
 - 518 • Ashish Patel, France Telecom
 - 519 • Greg Whitehead, Hewlett-Packard
 - 520 • Guy Denton, IBM
 - 521 • Heather Hinton, IBM
 - 522 • Anthony Nadalin, IBM
 - 523 • Eric Tiffany, IEEE Industry Standards and Technology Org (IEEE-ISTO)
 - 524 • Scott Cantor, Internet2
 - 525 • Bob Morgan, Internet2
 - 526 • Tom Scavo, National Center for Supercomputing Applications (NCSA)
 - 527 • Peter Davis, Neustar, Inc.
 - 528 • Jeff Hodges, Neustar, Inc.
 - 529 • Frederick Hirsch, Nokia Corporation
 - 530 • Abbie Barbir, Nortel Networks Limited
 - 531 • Paul Madsen, NTT Corporation
 - 532 • Ari Kermaier, Oracle Corporation
 - 533 • Prateek Mishra, Oracle Corporation
 - 534 • Brian Campbell, Ping Identity Corporation
 - 535 • Rob Philpott, RSA Security
 - 536 • Jahan Moreh, Sigaba Corp.
 - 537 • Bhavna Bhatnagar, Sun Microsystems
 - 538 • Eve Maler, Sun Microsystems
 - 539 • Emily Xu, Sun Microsystems
 - 540 • David Staggs, Veterans Health Administration

541

542

543 **Appendix B. Notices**

544 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that
545 might be claimed to pertain to the implementation or use of the technology described in this document or
546 the extent to which any license under such rights might or might not be available; neither does it represent
547 that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to
548 rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made
549 available for publication and any assurances of licenses to be made available, or the result of an attempt
550 made to obtain a general license or permission for the use of such proprietary rights by implementors or
551 users of this specification, can be obtained from the OASIS Executive Director.

552 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications,
553 or other proprietary rights which may cover technology that may be required to implement this
554 specification. Please address the information to the OASIS Executive Director.

555 | **Copyright © OASIS Open 2006Z. All Rights Reserved.**

556 This document and translations of it may be copied and furnished to others, and derivative works that
557 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published
558 and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice
559 and this paragraph are included on all such copies and derivative works. However, this document itself
560 may not be modified in any way, such as by removing the copyright notice or references to OASIS, except
561 as needed for the purpose of developing OASIS specifications, in which case the procedures for
562 copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to
563 translate it into languages other than English.

564 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors
565 or assigns.

566 This document and the information contained herein is provided on an "AS IS" basis and OASIS
567 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
568 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
569 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.