



1

2 **SAMLv2: HTTP POST “SimpleSign”**
3 **Binding**

4 **Draft, ~~8-29~~ September 2006_**

5 **Document identifier:**

6 | draft-hodges-saml-binding-simplesign-0~~2~~⁴

7 **Location:**

8

9 **Editors:**

10 | Jeff Hodges, NeuStar
11 | Scott Cantor, Internet2
12

13

14 **Abstract:**

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

20 **Status:**

21 This is an individually-authored working draft published to the Security Services Technical
22 Committee (SSTC/SAML), and has no official standing.
23 Committee members should submit comments to the security-services@lists.oasis-open.org list.
24 Non-committee members who wish to comment may do so on the [SAML-dev@lists.oasis-](mailto:SAML-dev@lists.oasis-open.org)
25 [open.org](mailto:SAML-dev@lists.oasis-open.org) mailing list (one must be a list subscriber to post. To subscribe, send mail to
26 <mailto:saml-dev-subscribe@lists.oasis-open.org>).
27

28 **Table of Contents**

29 1 Introduction..... 4

30 1.1 Protocol Binding Concepts..... 4

31 1.2 Notation..... 4

32 2 HTTP POST Binding - SimpleSign..... 6

33 1.2.1 Required Information..... 6

34 1.2.2 Overview..... 6

35 1.2.3 RelayState..... 6

36 1.2.4 Message Encoding and Conveyance..... 7

37 1.2.5 SimpleSign Signature..... 8

38 1.2.6 SimpleSign Signature Verification..... 8

39 1.2.7 Message Exchange..... 9

40 1.2.7.1 HTTP and Caching Considerations..... 11

41 1.2.7.2 Security Considerations..... 11

42 1.2.8 Error Reporting..... 11

43 1.2.9 Metadata Considerations..... 12

44 1.2.10 Note to Implementors..... 12

45 1.2.11 Example SAML Message Exchange Using HTTP POST - SimpleSign..... 12

46 3 References..... 15

47 Appendix B. Acknowledgments..... 17

48 Appendix B. Acknowledgments..... 17

49 Appendix C. Notices..... 18

50 Appendix C. Notices..... 18

1 Introduction

51

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

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

1.1 Protocol Binding Concepts

60

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

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

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

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

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

1.2 Notation

75

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

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

80 `Example code listings appear like this.`

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

83 Conventional XML namespace prefixes are used throughout this specification to stand for their respective
84 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].
SOAP-ENV:	http://schemas.xmlsoap.org/soap/envelope	This namespace is defined in SOAP V1.1

85

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

2 HTTP POST Binding--SimpleSign

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

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

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

1.2.1 Required Information

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

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

Description: Given below.

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

1.2.2 Overview

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

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

1.2.3 RelayState

RelayState data MAY be included with a SAML protocol message transmitted with this binding. The value MUST NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the [SAML protocol](#) message independent of any other protections that may or may not exist during message transmission. Signing is not realistic given the space limitation, but because the value is exposed to third-party tampering, the entity SHOULD ensure that the value has not been tampered with by using a checksum, a pseudo-random value, or similar means.

If a SAML request message is accompanied by RelayState data, then the SAML responder MUST return its SAML protocol response [message](#) using a binding that also supports a RelayState mechanism, and it MUST place the exact data it received with the request into the corresponding RelayState parameter in

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

134 1.2.4 Message Encoding and Conveyance

135 This section describes how to encode a SAML [protocol](#) message, and thus any SAML assertion(s) it may
136 contain, into HTML FORM “control(s)” [HTML401] (Section 17), thus enabling the SAML [protocol](#)
137 message to be conveyed via the HTTP POST method.

138 A SAML protocol message is form-encoded by:

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

152 SAML [protocol](#) messages, ~~MUST NOT and any SAML assertions contained within the SAML protocol~~
153 ~~messages, MAY be signed using [XMLSig], and if so, any such signatures MUST remain intact.~~

154 ~~**NOTE:** If a SAML message is so signed before being processed as defined herein, the~~
155 ~~SAML message's XML digital signature MUST be removed as described in section~~
156 ~~3.4.4.1 step 1.~~

157 ~~However, any SAML assertions contained within the message MAY be signed via~~

159 ~~Rather, if a SAML message is to be signed — which this binding leaves as a decision of the implementor~~
160 ~~and/or deployer — it MUST, and if so, any such signatures MUST remain intact. Additionally, SAML~~
161 ~~protocol messages MAY be signed using the technique given below in section 2.0.5. This technique is~~
162 ~~referred to as the “SimpleSign technique”. The resultant SimpleSign signature value is conveyed in a form~~
163 ~~control value named `Signature`, and the signature algorithm is conveyed in a form control value named~~
164 ~~`SigAlg`. These form control values are included in the form data set constructed in step 2 above.~~

165 If the [SAML protocol](#) message is signed [using SimpleSign](#), the `Destination` XML attribute in the root
166 SAML element of the SAML protocol message MUST contain the URL to which the sender has instructed
167 the user agent to deliver the message. The recipient MUST then verify that the value matches the location
168 at which the [SAML protocol](#) message has been received. Also, the signer's certificate or other keying
169 information MAY be included in a form control named `KeyInfo`. This form control, if present, MUST
170 contain a base-64 encoded `<ds:KeyInfo>` element ([XMLSig] base-64 [encodingeencoding](#) is done as in
171 step 1, above). This form control MUST NOT be included in the signed form data set constructed in step 2
172 above.

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

176 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using

177 | this binding to which the SAML [protocol](#) message is to be delivered. The `method` attribute MUST be
178 | "POST". The `enctype` attribute specifies the form content type and MUST be `application/x-www-`
179 | `form-urlencoded`.

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

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

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

191 | 1.2.5 [SimpleSign Signature](#)

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

193 | 1. The signature algorithm used MUST be identified by a URI, specified according to [XMLSig] or
194 | whatever specification governs the algorithm. The following signature algorithms (see [XMLSig])
195 | and their URI representations MUST be supported with this encoding mechanism:

- 196 | • DSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#dsa-sha1>
- 197 | • RSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#rsa-sha1>

199 | 2. A string consisting of the concatenation of the `RelayState` (if present), `SigAlg`, and
200 | `SAMLRequest` (or `SAMLResponse`) values (as appropriate), as defined in section 2.0.4 above, is
201 | constructed in one of the following ways (each individually ordered as shown):

```
202 | SAMLRequest=value&RelayState=value&SigAlg=value  
203 |  
204 | SAMLResponse=value&RelayState=value&SigAlg=value  
205 |
```

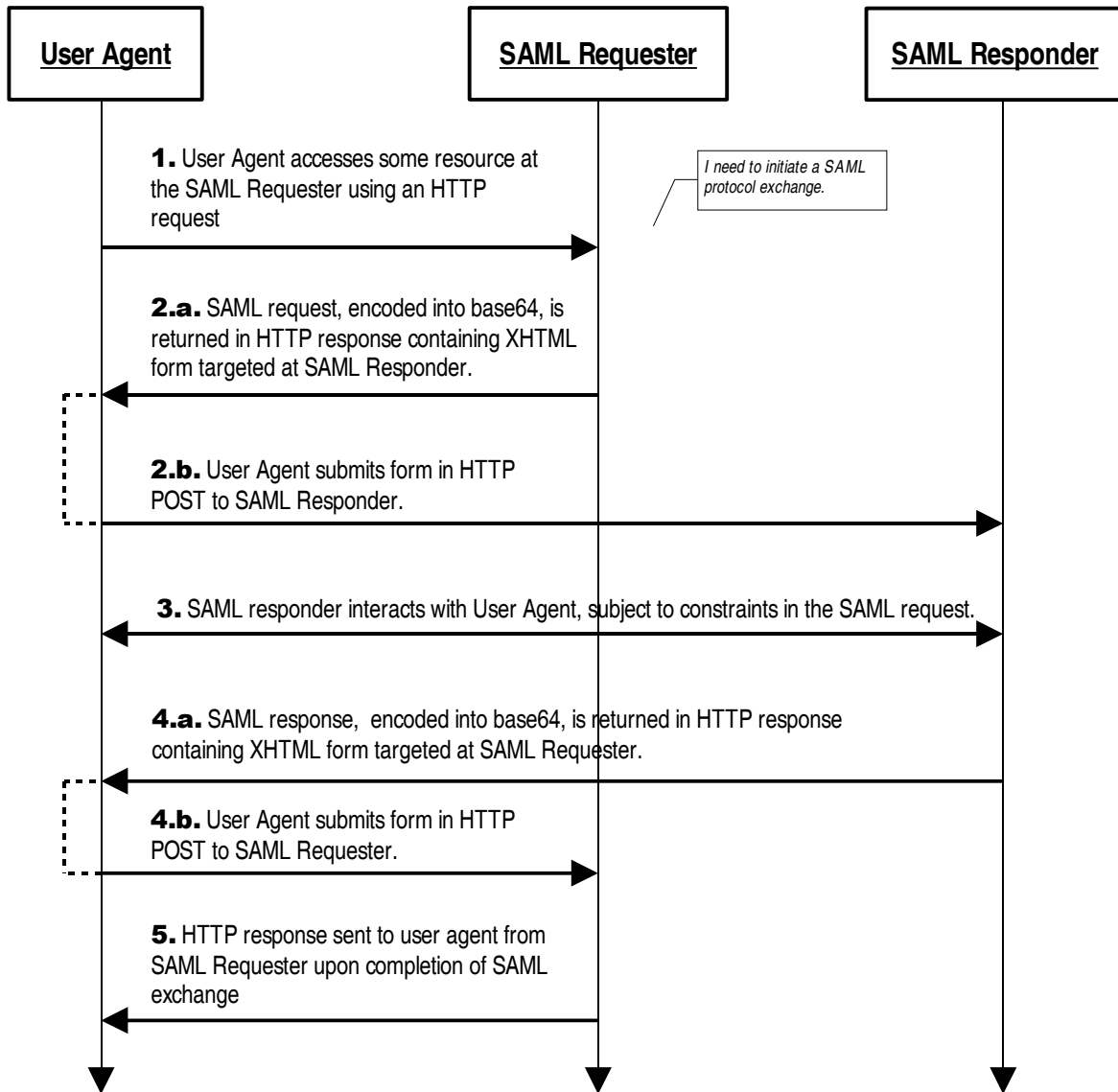
- 207 | 3. The resultant octet string is fed into the signature algorithm.
- 208 | 4. The value yielded by the signature algorithm is base64 encoded (see [RFC2045]), and used as the
209 | value for the `Signature` form control as discussed in section 2.0.4, above.

210 | 1.2.6 [SimpleSign Signature Verification](#)

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

220 **1.2.7 Message Exchange**

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



- 230 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
231 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 232 2. (a) The system entity acting as a SAML requester responds to an HTTP request from the user
233 agent by returning a SAML request. The request is returned in an XHTML document containing the
234 form and content defined in Section 2.0.4, above. (b) The user agent delivers the SAML request by
235 issuing an HTTP POST request to the SAML responder.
- 236 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
237 SAML response or it MAY return arbitrary content to facilitate subsequent interaction with the user
238 agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to
239 indicate the requester's level of willingness to permit this kind of interaction (for example, the
240 `IsPassive` attribute in `<samlp:AuthnRequest>` [SAMLCore]).
- 241 4. Eventually the responder SHOULD (a) return a SAML response to the user agent to be (b) returned
242 to the SAML requester. The SAML response is returned in the same fashion as described for the
243 SAML request in step 2, if this or a similar binding is employed for this step. Otherwise, details may
244 vary.

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

247 1.2.7.1 HTTP and Caching Considerations

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

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

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

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

254 1.2.7.2 Security Considerations

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

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

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

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

271 [If the SAML protocol messages are signed using \[XMLSig\] then the Destination XML attribute in the](#)
272 [root SAML element of the SAML protocol message MUST contain the URL to which the sender has](#)
273 [instructed the user agent to deliver the message. The recipient MUST then verify that the value matches](#)
274 [the location at which the message has been received.](#)

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

280 1.2.8 Error Reporting

281 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD
282 return a response message with a second-level `<samlp:StatusCode>` value of
283 `urn:oasis:names:tc:SAML:2.0:status:RequestDenied`.

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

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

288 1.2.9 Metadata Considerations

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

293 1.2.10 Note to Implementors

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

300 1.2.11 Example SAML Message Exchange Using HTTP POST--SimpleSign

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

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

```
306 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
307 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
308 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
309 21T19:00:49Z" Version="2.0">  
310 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
311 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
312 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
313 <samlp:SessionIndex>1</samlp:SessionIndex>  
314 </samlp:LogoutRequest>  
315  
316 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
317 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
318 ID="b0730d21b628110d8b7e004005b13a2b"  
319 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"  
320 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">  
321 <Issuer>https://ServiceProvider.com/SAML</Issuer>  
322 <samlp:Status>  
323 <samlp:StatusCode  
324 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>  
325 </samlp:Status>  
326 </samlp:LogoutResponse>  
327
```

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

```
331 HTTP/1.1 200 OK  
332 Date: 21 Jan 2004 07:00:49 GMT  
333 Content-Type: text/html; charset=iso-8859-1  
334  
335 <?xml version="1.0" encoding="UTF-8"?>
```

```

336 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
337 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
338 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
339 <body onload="document.forms[0].submit()">
340
341 <noscript>
342 <p>
343 <strong>Note:</strong> Since your browser does not support JavaScript,
344 you must press the Continue button once to proceed.
345 </p>
346 </noscript>
347
348 <form action="http://ServiceProvider.com/SAML/SLO/Browser" method="post">
349 <div>
350 <input type="hidden" name="RelayState"
351 value="0043bfc1bc45110dae17004005b13a2b"/>
352 <input type="hidden" name="SAMLRequest"
353 value="PHNhbWxwOkxvZ291dFJlcXVlc3QgeG1sbnM6c2FtbHA9InVybjpvYXNpczpuYW1l
354 czp0YzptQU1MOjIuMDpwcm90b2NvbCIgeG1sbnM9InVybjpvYXNpczpuYW1lc3p0
355 YzptQU1MOjIuMDphc3Nlc3Npb24iCiAgICBjRD0iZDZiN2MzODhjZWZmZmZhN2Mz
356 OWMYOGZkMjk4NjQ0YTgiIElzc3VlSW5zdGFudD0iMjAwNC0wMS0yMVQxOTowMDo0
357 OVoiiFZlcnNpb249IjIuMCI+CiAgICA8SXNzdWVYpMh0dHBzOi8vSWRlbnRpdHlQ
358 cm92aWRlci5jb20vU0FNTDdwvSXNzdWVYpGogICAgPE5hbWVJRjRlbnRpdHlQ
359 bjpvYXNpczpuYW1lc3p0YzptQU1MOjIuMDpwcm90b2NvbCIgeG1sbnM9InVybjpvYXNpczpuYW1l
360 bnRpdHlQcm92aWRlci5jb20vU0FNTDdwvSXNzdWVYpGogICAgPE5hbWVJRjRlbnRpdHlQ
361 PgogICAgPHNhbWxwO1Nlc3Npb25Jb3RlbnRpdHlQcm92aWRlci5jb20vU0FNTDdwvSXNzdWVYpGogICAgPE5hbWVJRjRlbnRpdHlQ
362 Cjwvc2FtbHA6TG9nb3V0UmVxdWVzdD4K"/>
363 <input type="hidden" name="Signature"
364 value="J4if7CCeHVfn4H6hMZN5fijOjQIyZ/laoFUZWz4LCRN3J82UeoyYvAiTD0QOUZHT
365 RJNU1lWGub1pw4QR9MH5bwfLEa8XDivA118dR0Q7YN5L/U5rmbxnglQ9pV0jt44c
366 RNeqtbLW0YF4plfcqg7E5iOSljE3QLkiaAdkAec2a4HwPFkn/JP7wO11Mc6kU8ML
367 CBbZAa3+94ZvVwHBEdyCdU+lyEvf+JGxTw66BwI2ugmPfxvoJdsOOAWwS3KhAFhL
368 LSPXnhb3nd/ovKNNV/khZYwqsFTFNTMA+0JraKsZiCrTEzZEPXaP9KilrjPIIvRV
369 xDQhETj96flk5zmkEM3ruw==" />
370 <input type="hidden" name="SigAlg"
371 value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
372 </div>
373 <noscript>
374 <div>
375 <input type="submit" value="Continue"/>
376 </div>
377 </noscript>
378 </form>
379 </body>
380 </html>
381

```

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

```

385 HTTP/1.1 200 OK
386 Date: 21 Jan 2004 07:00:49 GMT
387 Content-Type: text/html; charset=iso-8859-1
388
389 <?xml version="1.0" encoding="UTF-8"?>
390 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
391 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
392 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
393 <body onload="document.forms[0].submit()">
394
395 <noscript>
396 <p>

```

```

397 <strong>Note:</strong> Since your browser does not support JavaScript,
398 you must press the Continue button once to proceed.
399 </p>
400 </noscript>
401
402 <form action="https://IdentityProvider.com/SAML/SLO/Response"
403 method="post">
404 <div>
405 <input type="hidden" name="RelayState"
406 value="0043bfclbc45110dae17004005b13a2b"/>
407 <input type="hidden" name="SAMLResponse"
408 value="PHNhbWxwOkxvZ291dFJlcXVlc3QgeG1sbnM6c2FtbHA9InVybjpvYXNpczpuYW1l
409 czp0YzptQU1MOjIuMDpwcm90b2NvbCIgeG1sbnM9InVybjpvYXNpczpuYW1lc3p0
410 YzptQU1MOjIuMDphc3NlcnRpb24iCiAgICBjRD0iZDZiN2MzODhjZWZmNmZhN2Mz
411 OWMyOGZkMjk4NjQ0YTgiIElzc3VlSW5zdGFudD0iMjAwNC0wMS0yMVQxOTowMDo0
412 OVoiIFZlcnNpb249IjIuMCI+CiAgICAgICA8SjNzdWVpPmh0dHBzOi8vSWRlbnRpdHlQ
413 cm92aWRlci5jb20vU0FNTDdwSXNzdWVpPgogICAgPE5hbWVJRCBGb3JtYXQ9InVy
414 bjpvYXNpczpuYW1lc3p0YzptQU1MOjIuMDpuYW1laWQtZm9ybWF0OnBlcnNpc3Rl
415 bnQiPjAwNWUwNmUwLWVfKODItMTEwZC1hNTU2LTAwNDAwNW1xM2EyYjwvTmFtZU1E
416 PgogICAgPHNhbWxwOlNlc3Npb25JbMtleD4xPC9zYW1scDpTZXRzaW9uSW5kZXg+
417 Cjwvc2FtbHA6TG9nb3V0UmVxdWVzdD4K"/>
418 <input type="hidden" name="Signature"
419 value="DCDqAwIDqSwyXGvG2cYvNjmj7P1kt0+kbCfRjq9gGTrN4KKPqvQl5EsFrWRkM0dx
420 xuwPldWPKvfgX6rt+pKwLgCt1TqRj+71y+VdGS8ORsBeEIURRn9wSu+pKsWiHexw
421 KnIe65bjONbg2db44QOWZlDe76fLi05Psy/7HZTQuMoDRFYSR//VyNGHQmf9Sxi6
422 mkmrYMXPOyZAUfNhX4eLaXFfwCHt0yRrEcm/PAEDDa7uqe8Uo5ilstgXDWDodWdk
423 Szk8ZS1irjFkvtxH7FJlm9ADt1W/SoX92jGjMIrdQwCyArI6o8KtiDp/cjDjHZGi
424 XLx2WvS7GEibA7Qd+5hSBQ==" />
425 <input type="hidden" name="SigAlg"
426 value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
427 </div>
428 <noscript>
429 <div>
430 <input type="submit" value="Continue"/>
431 </div>
432 </noscript>
433 </form>
434 </body>
435 </html>

```

3 References

436

- 437 [HTML401] D. Raggett et al. *HTML 4.01 Specification*. World Wide Web Consortium
438 Recommendation, December 1999. See <http://www.w3.org/TR/html4>.
- 439 [RFC2045] N. Freed et al. *Multipurpose Internet Mail Extensions (MIME) Part One: Format
440 of Internet Message Bodies*, IETF RFC 2045, November 1996. See
441 <http://www.ietf.org/rfc/rfc2045.txt>.
- 442 [RFC2119] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF
443 RFC 2119, March 1997. See <http://www.ietf.org/rfc/rfc2119.txt>.
- 444 [RFC2246] T. Dierks et al. *The TLS Protocol Version 1.0*. IETF RFC 2246, January 1999.
445 See <http://www.ietf.org/rfc/rfc2246.txt>.
- 446 [RFC2616] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC 2616, June
447 1999. See <http://www.ietf.org/rfc/rfc2616.txt>.
- 448 [SAMLBind] S. Cantor et al. *Bindings for the OASIS Security Assertion Markup Language
449 (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-bindings-2.0-os.
450 See <http://www.oasis-open.org/committees/security/>.
- 451 [SAMLCore] S. Cantor et al. *Assertions and Protocols for the OASIS Security Assertion
452 Markup Language (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-
453 core-2.0-os. See <http://www.oasis-open.org/committees/security/>.
- 454 [SAMLGloss] J. Hodges et al. *Glossary for the OASIS Security Assertion Markup Language
455 (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-glossary-2.0-os.
456 See <http://www.oasis-open.org/committees/security/>.
- 457 [SAMLMeta] S. Cantor et al. *Metadata for the OASIS Security Assertion Markup Language
458 (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-metadata-2.0-os.
459 See <http://www.oasis-open.org/committees/security/>.
- 460 [SAMLProf] S. Cantor et al. *Profiles for the OASIS Security Assertion Markup Language
461 (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-profiles-2.0-os. See
462 <http://www.oasis-open.org/committees/security/>.
- 463 [SAMLSecure] F. Hirsch et al. *Security and Privacy Considerations for the OASIS Security
464 Assertion Markup Language (SAML) V2.0*. OASIS SSTC, March 2005.
465 Document ID saml-sec-consider-2.0-os. See [http://www.oasis-
open.org/committees/security/](http://www.oasis-
466 open.org/committees/security/).
- 467 [SOAP11] D. Box et al. *Simple Object Access Protocol (SOAP) 1.1*. World Wide Web
468 Consortium Note, May 2000. See [http://www.w3.org/TR/2000/NOTE-SOAP-
20000508/](http://www.w3.org/TR/2000/NOTE-SOAP-
469 20000508/).
- 470 [SSL3] A. Frier et al. *The SSL 3.0 Protocol*. Netscape Communications Corp, November
471 1996.
- 472 [SSTCWeb] OASIS Security Services Technical Committee website, [http://www.oasis-
open.org/committees/security](http://www.oasis-
473 open.org/committees/security).
- 474 [XHTML] *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*.
475 World Wide Web Consortium Recommendation, August 2002. See
476 <http://www.w3.org/TR/xhtml1/>.
- 477 [XMLSig] D. Eastlake et al. *XML-Signature Syntax and Processing*. World Wide Web
478 Consortium Recommendation, February 2002. See
479 <http://www.w3.org/TR/xmlsig-core/>.

481 **Appendix A. Acknowledgments**

482 @@TODO: update as appropriate

483 The editors acknowledge the contributions of the OASIS Security Services Technical Committee, whose
484 voting members at the time of publication were:

- 485 • TBD

486

487 The editors also acknowledge the following former SSTC members for their contributions to this or
488 previous versions of the OASIS Security Assertions Markup Language Standard:

- 489 • TBD

490

491

492 Appendix B. Notices

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

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

504 **Copyright © OASIS Open 2005. All Rights Reserved.**

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

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

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