



SAMLv2.0 HTTP POST “SimpleSign” Binding

Draft 02, 5 January 2007

Document identifier:

draft-sstc-saml-binding-simplesign-02

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 **Working Draft** 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>).

34	Table of Contents	
35	1 Introduction.....	3
36	1.1 Protocol Binding Concepts.....	3
37	1.2 Notation.....	3
38	2 HTTP POST Binding-SimpleSign.....	5
39	2.1 Required Information.....	5
40	2.2 Overview.....	5
41	2.3 Relay State.....	5
42	2.4 Message Encoding and Conveyance.....	6
43	2.5 SimpleSign Signature.....	7
44	2.6 SimpleSign Signature Verification.....	7
45	2.7 Message Exchange.....	8
46	2.7.1 HTTP and Caching Considerations.....	10
47	2.7.2 Security Considerations.....	10
48	2.8 Error Reporting.....	10
49	2.9 Metadata Considerations.....	11
50	2.10 Note to Implementors.....	11
51	2.11 Example.....	11
52	3 References.....	14
53	Appendix A. Acknowledgments.....	15
54	Appendix B. Notices.....	16

1 Introduction

55

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

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

1.1 Protocol Binding Concepts

64

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

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

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

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

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

1.2 Notation

79

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

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

84 `Example code listings appear like this.`

85 **Note:** Notes like this are sometimes used to highlight non-normative commentary.
86

87 Conventional XML namespace prefixes are used throughout this specification to stand for their respective
88 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].

89

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

93 2 HTTP POST Binding-SimpleSign

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

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

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

107 2.1 Required Information

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

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

110 **Description:** Given below.

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

112 2.2 Overview

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

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

124 2.3 Relay State

125 RelayState data MAY be included with a SAML protocol message transmitted with this binding. The value
126 MUST NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the
127 message, either via a digital signature (see section 2.5) or by some independent means.

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

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

135 2.4 Message Encoding and Conveyance

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

139 A SAML protocol message is form-encoded by:

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

153 SAML protocol messages, and any SAML assertions contained within the SAML protocol messages,
154 MAY be signed using [XMLSig], and if so, any such signatures MUST remain intact. Additionally, SAML
155 protocol messages MAY be signed using the technique given below in section 2.5. This technique is
156 referred to as the “SimpleSign technique”. The SimpleSign signature value is conveyed in a form control
157 value named `Signature`, and the signature algorithm is conveyed in a form control value named
158 `SigAlg`. These form control values are included in the form data set constructed in step 2 above.

159 If the SAML protocol message is signed using SimpleSign, the `Destination` XML attribute in the root
160 SAML element of the SAML protocol message MUST contain the URL to which the sender has instructed
161 the user agent to deliver the message. The recipient MUST then verify that the value matches the location
162 at which the SAML protocol message has been received. Also, the signer’s certificate or other keying
163 information MAY be included in a form control named `KeyInfo`. This form control, if present, MUST
164 contain a base-64 encoded `<ds:KeyInfo>` element [XMLSig] (base-64 encoding is done as in step 1,
165 above).

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

169 The `action` attribute of the form MUST be the recipient’s HTTP endpoint for the protocol or profile using
170 this binding to which the SAML protocol message is to be delivered. The `method` attribute MUST be
171 “POST”. The `enctype` attribute specifies the form content type and MUST be `application/x-www-`
172 `form-urlencoded`.

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

176 Any technique supported by the user agent MAY be used to cause the submission of the form (to cause it
177 to be conveyed to the SAML protocol message recipient), and any form content necessary to support this

178 MAY be included, such as submit controls and client-side scripting commands. However, the recipient
179 MUST be able to process the message without regard for the mechanism by which the form submission is
180 initiated.

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

184 2.5 SimpleSign Signature

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

- 186 1. The signature algorithm used MUST be identified by a URI, specified according to [XMLSig] or
187 whatever specification governs the algorithm. The following signature algorithms (see [XMLSig])
188 and their URI representations MUST be supported with this encoding mechanism:
 - 189 • DSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#dsa-sha1>
 - 190 • RSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#rsa-sha1>
- 192 2. A string consisting of the concatenation of the raw, unencoded XML making up the SAML protocol
193 message (NOT the base64-encoded version), the `RelayState` value (if present), and the
194 `SigAlg` value, is constructed in one of the following ways (each individually ordered as shown):

```
195 SAMLRequest=value&RelayState=value&SigAlg=value  
196  
197 SAMLResponse=value&RelayState=value&SigAlg=value  
198
```

- 200 3. The resultant octet string is fed into the signature algorithm.
- 201 4. The value yielded by the signature algorithm is base64 encoded (see [RFC2045]), and used as the
202 value for the `Signature` form control as discussed in section 2.4, above.

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

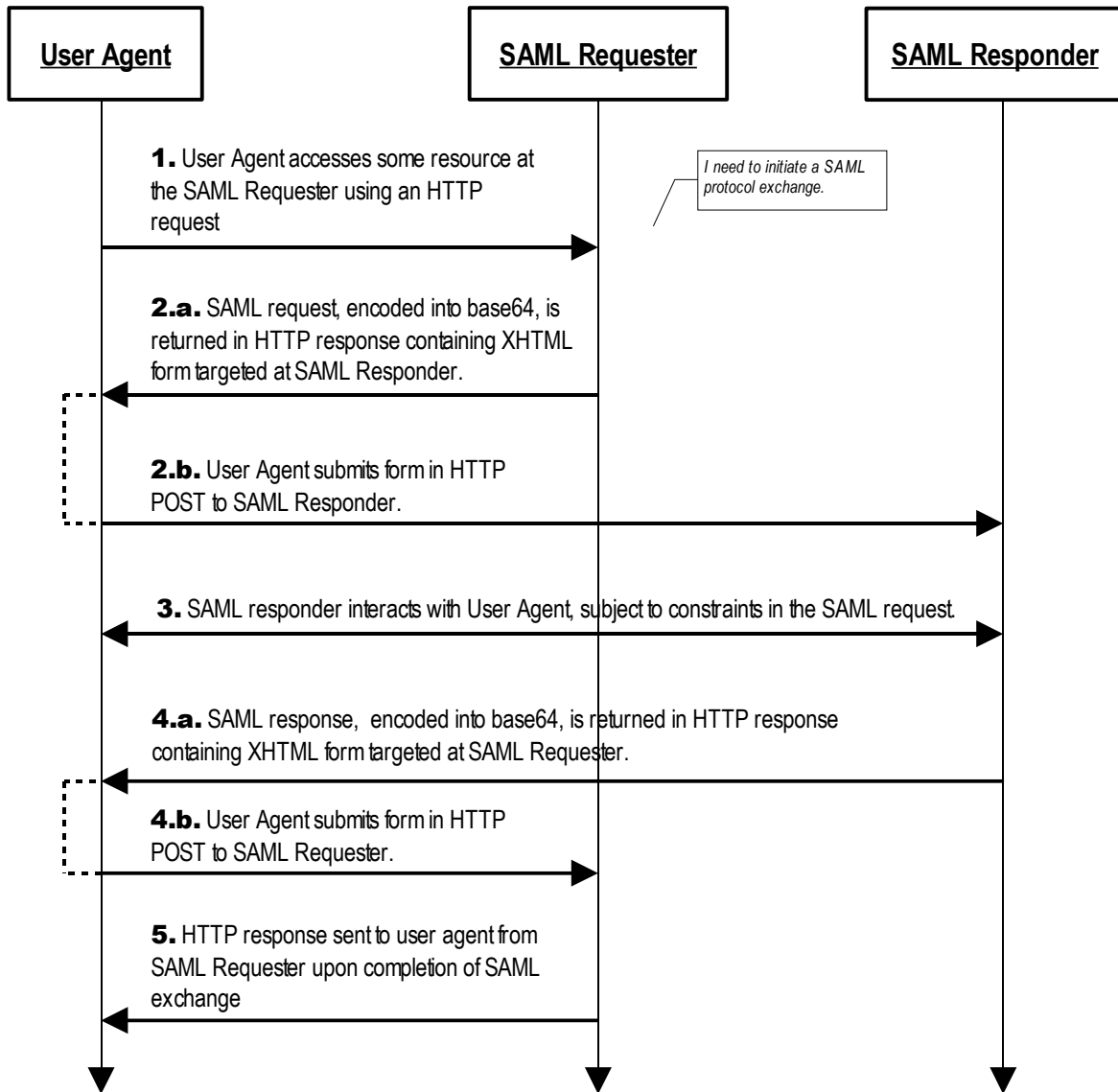
208 The original XML MUST be concatenated with the other information as shown above without regard for
209 any embedded whitespace, even if the result spans multiple lines. The specific whitespace characters
210 present will be safely encoded in base64 and then recovered by the relying party for use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

212 2.6 SimpleSign Signature Verification

213 To verify a received SAML protocol message, which was signed using SimpleSign and conveyed by this
214 binding, the receiver MUST extract the form control values for the `RelayState` (if present), `SigAlg`, and
215 `SAMLRequest` (or `SAMLResponse`) values (as appropriate) from the received HTTP message. Then the
216 receiver reconstructs the string as described in section 2.5 step 2, above. The signature value conveyed
217 in the `Signature` control value is then checked against this string per the signature algorithm given by
218 the `SigAlg` control value, and using (as appropriate, see [XMLSig]) the keying material obtained via the
219 `<ds:KeyInfo>` conveyed in the `KeyInfo` control value (if present). Error handling and generated
220 messages as a result of the signature not verifying are implementation-dependent.

221 **2.7 Message Exchange**

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



- 231 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
232 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 233 2. (a) The system entity acting as a SAML requester responds to an HTTP request from the user
234 agent by returning a SAML request. The request is returned in an XHTML document containing the
235 form and content defined in Section 2.4, above. (b) The user agent delivers the SAML request by
236 issuing an HTTP POST request to the SAML responder.
- 237 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
238 SAML response or it MAY return arbitrary content to facilitate subsequent interaction with the user
239 agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to
240 indicate the requester's level of willingness to permit this kind of interaction (for example, the
241 `IsPassive` attribute in `<samlp:AuthnRequest>` [SAMLCore]).
- 242 4. Eventually the responder SHOULD (a) return a SAML response to the user agent to be (b) returned
243 to the SAML requester. The SAML response is returned in the same fashion as described for the
244 SAML request in step 2, if this or a similar binding is employed for this step. Otherwise, details may
245 vary.

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

248 2.7.1 HTTP and Caching Considerations

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

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

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

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

255 2.7.2 Security Considerations

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

261 This binding SHOULD NOT be used if the content of the request or response should not be exposed to
262 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is
263 OPTIONAL and depends on the environment of use. If on-the-wire confidentiality is necessary, SSL 3.0
264 [SSL3] or TLS 1.0 [RFC2246] SHOULD be used to protect the overall HTTP messages, and the conveyed
265 SAML protocol messages, in transit between the user agent and the SAML requester and 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 the SimpleSign approach or [XMLSig]) then the
272 `Destination` XML attribute in the root SAML element of the SAML protocol message MUST contain the
273 URL to which the sender has instructed the user agent to deliver the message. The recipient MUST then
274 verify that the value matches 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 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

288

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 2.1 is used as the value for the `Binding` attribute of any endpoint elements.

293

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

2.11 Example

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 2.5, 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 czp0YzptQU1MOjIuMDpwcm90b2NvbCIgeG1sbnM9InVybjpvYXNpczpuYW1lczp0
355 YzptQU1MOjIuMDphc3Nlc3Rpb24iCiAgICBjRD0iZDZiN2MzODhjZWZmZmZmZmN2Mz
356 OWMYOGZkMjk4NjQ0YTgiIElzc3VlSW5zdGFudD0iMjAwMDAwMS0yMVQxOTowMDQ0
357 OVOiIFZlcnNpb249IjIuMCI+CiAgICAgICA8SXXNdWVybWVybWVybWVybWVybWVybW
358 cm92aWRlcjE5bjI0vU0FNTDdWvSXNzdWVvPgogICAgPE5hbWVwJRCBGB3JtYXQ9InVy
359 bjpvYXNpczpuYW1lczp0YzptQU1MOjIuMDpueW1laWQvZm9ybnR3bWV0bnR1cnR3Rl
360 bnQiPjAwNWVwNmUwLWFkODI1MTEwZC1hNTU2LTAwNDAwNW1xM2EYyYyYyYyYyYyYyYyYy
361 PgogICAgPHNhbWxwO1Nlc3Npb25JbmlleD4xPC9zYW1scDpTZXNzaW9uSW5kZXZgc+
362 Cjwvc2FtbHA6TG9nb3V0UmVxdWVzdD4K"/>
363 <input type="hidden" name="Signature"
364 value="J4if7CCeHVFfn4H6hMZN5fijOjQIyZ/laoFUZWz4LCRN3J82UeoyYvAiTDoQOUZHT
365 RJNU1lWGublpw4QR9MH5bwfLEa8XDivA118dR0Q7YN5L/U5rmbxnglQ9pV0jt44c
366 RNeqtbLW0Yf4plfcqg7E5iOSlJE3QLkiaAdkAec2a4HwPFkn/JP7wO11Mc6kU8ML
367 CBbZaA3+94ZvVwHBEdyCdU+1yEvf+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+CiAgICA8SjNzdWVpPmh0dHBzOi8vSWRlbnRpdHlQ  
413 cm92aWRlci5jb20vU0FNTDdwSXNzdWVpPgogICAgPE5hbWVJRCBGb3JtYXQ9InVy  
414 bjpvYXNpczpuYW1lc3p0YzptQU1MOjIuMDpuYW1laWQtZm9ybWF0OnBlcnNpc3Rl  
415 bnQiPjAwNWewNmUwLWFkODItMTEwZC1hNTU2LTAwNDAwNW1xM2EyYjwvTmFtZU1E  
416 PgogICAgPHNhbWxwOlNlc3Npb25JbmlleD4xPC9zYW1scDpTZjNzaW9uSW5kZXg+  
417 Cjwvc2FtbHA6TG9nb3V0UmVxdWVzdD4K"/>  
418 <input type="hidden" name="Signature"  
419 value="DCDqAwIDqSwyXGvG2cYvNjmj7P1kt0+kbCfRjq9gGTrN4KKPqvQl5EsFrWRkMOdx  
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/>.

480 Appendix A. Acknowledgments

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

- 483
- 484 • Christopher Laskowski, Booz Allen Hamilton
 - 485 • Rebekah Metz, Booz Allen Hamilton
 - 486 • Hal Lockhart, BEA Systems, Inc.
 - 487 • Steve Anderson, BMC Software
 - 488 • Sharon Boeyen, Entrust
 - 489 • Thomas Wisniewski, Entrust
 - 490 • Carolina Canales-Valenzuela, Ericsson
 - 491 • Dana Kaufman, Forum Systems, Inc.
 - 492 • Ashish Patel, France Telecom
 - 493 • Greg Whitehead, Hewlett-Packard
 - 494 • Guy Denton, IBM
 - 495 • Heather Hinton, IBM
 - 496 • Anthony Nadalin, IBM
 - 497 • Eric Tiffany, IEEE Industry Standards and Technology Org (IEEE-ISTO)
 - 498 • Scott Cantor, Internet2
 - 499 • Bob Morgan, Internet2
 - 500 • Tom Scavo, National Center for Supercomputing Applications (NCSA)
 - 501 • Peter Davis, Neustar, Inc.
 - 502 • Jeff Hodges, Neustar, Inc.
 - 503 • Frederick Hirsch, Nokia Corporation
 - 504 • Abbie Barbir, Nortel Networks Limited
 - 505 • Paul Madsen, NTT Corporation
 - 506 • Ari Kermaier, Oracle Corporation
 - 507 • Prateek Mishra, Oracle Corporation
 - 508 • Brian Campbell, Ping Identity Corporation
 - 509 • Rob Philpott, RSA Security
 - 510 • Jahan Moreh, Sigaba Corp.
 - 511 • Bhavna Bhatnagar, Sun Microsystems
 - 512 • Eve Maler, Sun Microsystems
 - 513 • Emily Xu, Sun Microsystems
 - 514 • David Staggs, Veterans Health Administration

515

516

517 **Appendix B. Notices**

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

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

529 **Copyright © OASIS Open 2007. All Rights Reserved.**

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

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

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