



SAMLv2.0 HTTP POST “SimpleSign” Binding

Committee Draft 03 11 March 2008

Specification URIs:

This Version:

<http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-binding-simplesign-cd-03.html>

<http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-binding-simplesign-cd-03.odt>

<http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-binding-simplesign-cd-03.pdf>

Previous Version:

<http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-binding-simplesign-cd-02.html>

<http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-binding-simplesign-cd-02.odt>

<http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-binding-simplesign-cd-02.pdf>

Latest Version:

<http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-binding-simplesign.html>

<http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-binding-simplesign.odt>

<http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-binding-simplesign.pdf>

Technical Committee:

OASIS Security Services TC

Chairs:

Hal Lockhart, BEA Systems, Inc.

Prateek Mishra, Oracle Corporation

Editors:

Jeff Hodges, NeuStar

Scott Cantor, Internet2

Related Work:

This specification is an addition to the bindings described in the SAML V2.0 Bindings specification [SAMLBind].

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:

38 This document was last revised or approved by the SSTC on the above date. The level of
39 approval is also listed above. Check the current location noted above for possible later revisions
40 of this document. This document is updated periodically on no particular schedule.

41 TC members should send comments on this specification to the TC's email list.
42 Others should send comments to the TC by using the "Send A Comment" button on
43 the TC's web page at <http://www.oasis-open.org/committees/security>.

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

47 The non-normative errata page for this specification is located at [http://www.oasis-](http://www.oasis-open.org/committees/security)
48 [open.org/committees/security](http://www.oasis-open.org/committees/security).

Notices

50 Copyright © OASIS Open 2008. All Rights Reserved.

51 All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual
52 Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

53 This document and translations of it may be copied and furnished to others, and derivative works that
54 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published,
55 and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice
56 and this section are included on all such copies and derivative works. However, this document itself may
57 not be modified in any way, including by removing the copyright notice or references to OASIS, except as
58 needed for the purpose of developing any document or deliverable produced by an OASIS Technical
59 Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must
60 be followed) or as required to translate it into languages other than English.

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

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

68 OASIS requests that any OASIS Party or any other party that believes it has patent claims that would
69 necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard,
70 to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to
71 such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that
72 produced this specification.

73 OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of
74 any patent claims that would necessarily be infringed by implementations of this specification by a patent
75 holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR
76 Mode of the OASIS Technical Committee that produced this specification. OASIS may include such
77 claims on its website, but disclaims any obligation to do so.

78 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that
79 might be claimed to pertain to the implementation or use of the technology described in this document or
80 the extent to which any license under such rights might or might not be available; neither does it represent
81 that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to
82 rights in any document or deliverable produced by an OASIS Technical Committee can be found on the
83 OASIS website. Copies of claims of rights made available for publication and any assurances of licenses
84 to be made available, or the result of an attempt made to obtain a general license or permission for the
85 use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS
86 Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any
87 information or list of intellectual property rights will at any time be complete, or that any claims in such list
88 are, in fact, Essential Claims.

89 The name "OASIS" is a trademark of [OASIS](http://www.oasis-open.org), the owner and developer of this specification, and should be
90 used only to refer to the organization and its official outputs. OASIS welcomes reference to, and
91 implementation and use of, specifications, while reserving the right to enforce its marks against
92 misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

93 Table of Contents

94	1 Introduction.....	5
95	1.1 Protocol Binding Concepts.....	5
96	1.2 Notation.....	5
97	1.3 Normative References.....	6
98	1.4 Conformance.....	7
99	1.4.1 HTTP POST-SimpleSign Binding.....	7
100	2 HTTP POST-SimpleSign Binding.....	8
101	2.1 Required Information.....	8
102	2.2 Overview.....	8
103	2.3 Relay State.....	8
104	2.4 Message Encoding and Conveyance.....	9
105	2.5 SimpleSign Signature.....	10
106	2.6 SimpleSign Signature Verification.....	10
107	2.7 Message Exchange.....	11
108	2.7.1 HTTP and Caching Considerations.....	12
109	2.7.2 Security Considerations.....	12
110	2.8 Error Reporting.....	13
111	2.9 Metadata Considerations.....	13
112	2.10 Note to Implementors.....	13
113	2.11 Example.....	13
114	Appendix A. Acknowledgments.....	16

1 Introduction

115

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

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

1.1 Protocol Binding Concepts

124

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

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

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

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

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

1.2 Notation

139

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

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

144

145 `Example code listings appear like this.`

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

147 Conventional XML namespace prefixes are used throughout this specification to stand for their respective
148 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].

149

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

153 1.3 Normative References

- 154 **[HTML401]** D. Raggett et al. *HTML 4.01 Specification*. World Wide Web Consortium
 155 Recommendation, December 1999. See <http://www.w3.org/TR/html4>.
- 156 **[RFC2045]** N. Freed et al. *Multipurpose Internet Mail Extensions (MIME) Part One: Format
 157 of Internet Message Bodies*, IETF RFC 2045, November 1996. See
 158 <http://www.ietf.org/rfc/rfc2045.txt>.
- 159 **[RFC2119]** S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF
 160 RFC 2119, March 1997. See <http://www.ietf.org/rfc/rfc2119.txt>.
- 161 **[RFC2246]** T. Dierks et al. *The TLS Protocol Version 1.0*. IETF RFC 2246, January 1999.
 162 See <http://www.ietf.org/rfc/rfc2246.txt>.
- 163 **[RFC2616]** R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC 2616, June
 164 1999. See <http://www.ietf.org/rfc/rfc2616.txt>.
- 165 **[SAMLBind]** S. Cantor et al. *Bindings for the OASIS Security Assertion Markup Language
 166 (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-bindings-2.0-os.
 167 See <http://www.oasis-open.org/committees/security/>.
- 168 **[SAMLCore]** S. Cantor et al. *Assertions and Protocols for the OASIS Security Assertion
 169 Markup Language (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-
 170 core-2.0-os. See <http://www.oasis-open.org/committees/security/>.
- 171 **[SAMLGloss]** J. Hodges et al. *Glossary for the OASIS Security Assertion Markup Language
 172 (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-glossary-2.0-os.
 173 See <http://www.oasis-open.org/committees/security/>.
- 174 **[SAMLMeta]** S. Cantor et al. *Metadata for the OASIS Security Assertion Markup Language
 175 (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-metadata-2.0-os.
 176 See <http://www.oasis-open.org/committees/security/>.
- 177 **[SAMLProf]** S. Cantor et al. *Profiles for the OASIS Security Assertion Markup Language
 178 (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-profiles-2.0-os. See
 179 <http://www.oasis-open.org/committees/security/>.
- 180 **[SAMLSecure]** F. Hirsch et al. *Security and Privacy Considerations for the OASIS Security
 181 Assertion Markup Language (SAML) V2.0*. OASIS SSTC, March 2005.
 182 Document ID saml-sec-consider-2.0-os. See [http://www.oasis-
 open.org/committees/security/](http://www.oasis-

 183 open.org/committees/security/).
- 184 **[SOAP11]** D. Box et al. *Simple Object Access Protocol (SOAP) 1.1*. World Wide Web
 185 Consortium Note, May 2000. See [http://www.w3.org/TR/2000/NOTE-SOAP-
 20000508/](http://www.w3.org/TR/2000/NOTE-SOAP-

 186 20000508/).
- 187 **[SSL3]** A. Frier et al. *The SSL 3.0 Protocol*. Netscape Communications Corp, November
 188 1996.
- 189 **[SSTCWeb]** OASIS Security Services Technical Committee website, [http://www.oasis-
 open.org/committees/security/](http://www.oasis-

 190 open.org/committees/security/).
- 191 **[XHTML]** *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*.
 192 World Wide Web Consortium Recommendation, August 2002. See
 193 <http://www.w3.org/TR/xhtml1/>.

194 **[XMLSig]** D. Eastlake et al. *XML-Signature Syntax and Processing*. World Wide Web
195 Consortium Recommendation, February 2002. See
196 <http://www.w3.org/TR/xmlsig-core/>.

197 **1.4 Conformance**

198 **1.4.1 HTTP POST-SimpleSign Binding**

199 An implementation shall be considered conforming if it conforms to all normative requirements of section
200 2.

201 **2 HTTP POST-SimpleSign Binding**

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

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

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

215 **2.1 Required Information**

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

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

218 **Description:** Given below.

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

220 **2.2 Overview**

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

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

232 **2.3 Relay State**

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

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

240

241

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

245 2.4 Message Encoding and Conveyance

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

249 A SAML protocol message is form-encoded by:

- 250 1. Applying the base-64 encoding rules to the XML representation of the message. The resulting
251 base64-encoded value MAY be line-wrapped at a reasonable length in accordance with common
252 practice.
- 253 2. Encoding the result from the prior step into a "form data set", in the same fashion as is specified for
254 "successful controls" in [HTML401] (Section 17.13.3), as a form "control value". The HTML
255 document also MUST adhere to the XHTML specification, [XHTML].
 - 256 a. If the SAML protocol message is a SAML request, then the form "control name" used to convey
257 the SAML protocol message itself MUST be `SAMLRequest`.
 - 258 b. If the SAML protocol message is a SAML response, then the form "control name" used to
259 convey the SAML protocol message itself MUST be `SAMLResponse`.
 - 260 c. Any additional form controls or presentation, other than those noted below for including a
261 signature, MAY be included but MUST NOT be required in order for the recipient to nominally
262 process the SAML protocol message itself.

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

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

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

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

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

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

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

294 2.5 SimpleSign Signature

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

- 296 1. The signature algorithm used MUST be identified by a URI, specified according to [XMLSig] or
297 whatever specification governs the algorithm. The following signature algorithms (see [XMLSig])
298 and their URI representations MUST be supported with this encoding mechanism:
 - 299 • DSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#dsa-sha1>
 - 300 • RSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#rsa-sha1>
- 302 2. A string consisting of the concatenation of the raw, unencoded XML making up the SAML protocol
303 message (NOT the base64-encoded version), the `RelayState` value (if present), and the
304 `SigAlg` value, is constructed in one of the following ways (each individually ordered as shown):
305
306

```
SAMLRequest=value&RelayState=value&SigAlg=value
```


307

```
SAMLResponse=value&RelayState=value&SigAlg=value
```


308
- 310 3. The resultant octet string is fed into the signature algorithm.
- 311 4. The value yielded by the signature algorithm is base64 encoded (see [RFC2045]), and used as the
312 value for the `Signature` form control as discussed in section 2.4, above.

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

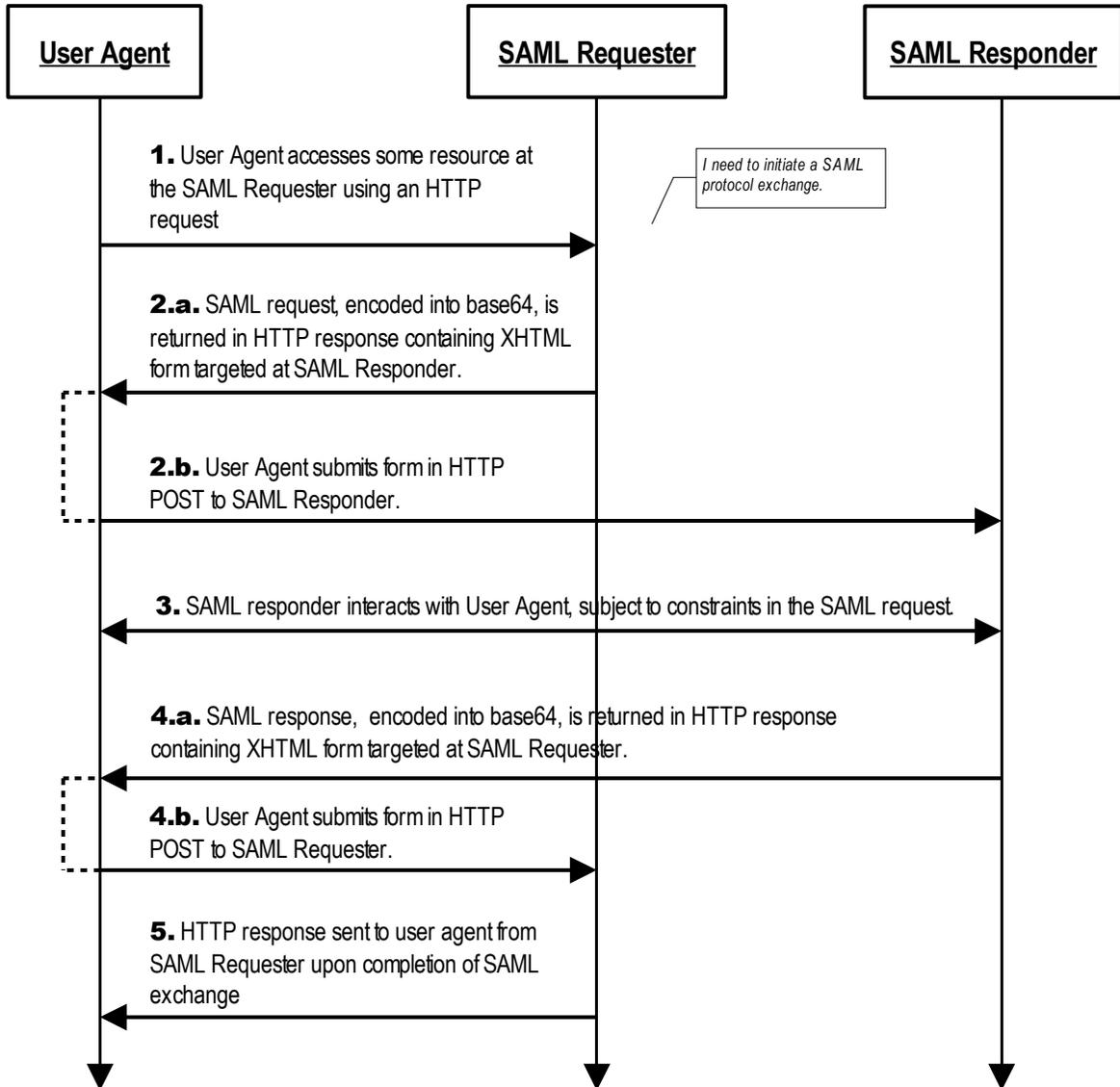
318 The original XML MUST be concatenated with the other information as shown above without regard for
319 any embedded whitespace, even if the result spans multiple lines. The specific whitespace characters
320 present will be safely encoded in base64 and then recovered by the relying party for use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

322 2.6 SimpleSign Signature Verification

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

331 **2.7 Message Exchange**

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



- 340 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
341 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 342 2. (a) The system entity acting as a SAML requester responds to an HTTP request from the user
343 agent by returning a SAML request. The request is returned in an XHTML document containing
344 the form and content defined in Section 2.4, above. (b) The user agent delivers the SAML request
345 by issuing an HTTP POST request to the SAML responder.
- 346 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
347 SAML response or it MAY return arbitrary content to facilitate subsequent interaction with the
348 user agent necessary to fulfill the request. Specific protocols and profiles may include
349 mechanisms to indicate the requester's level of willingness to permit this kind of interaction (for
350 example, the `IsPassive` attribute in `<samlp:AuthnRequest>` [SAMLCore]).
- 351 4. Eventually the responder SHOULD (a) return a SAML response to the user agent to be (b)
352 returned to the SAML requester. The SAML response is returned in the same fashion as
353 described for the SAML request in step 2, if this or a similar binding is employed for this step.
354 Otherwise, details may vary.
- 355 5. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to
356 the user agent.

357 2.7.1 HTTP and Caching Considerations

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

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

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

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

364 2.7.2 Security Considerations

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

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

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

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

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

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

389 2.8 Error Reporting

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

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

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

397 2.9 Metadata Considerations

398 Support for the HTTP POST-SimpleSign binding SHOULD be reflected by indicating URL endpoints at
399 which requests and responses for a particular protocol or profile should be sent. Either a single endpoint
400 or distinct request and response endpoints MAY be supplied [SAMLMeta]. The identification URI given in
401 section 2.1 is used as the value for the `Binding` attribute of any endpoint elements.

402 2.10 Note to Implementors

403 SAML protocol message recipients can distinguish between HTTP-SAML messages constructed via this
404 specification's HTTP POST-SimpleSign binding and ones constructed via the HTTP POST binding
405 [SAMLBind] by examining received HTTP messages for an XHTML form field with a name attribute value
406 of `Signature`. If this is present, then the message MUST be processed in accordance with this
407 specification. If not present, then the HTTP message MAY be processed in accordance with the HTTP
408 POST binding specification.

409 2.11 Example

410 In this example, a `<LogoutRequest>` and `<LogoutResponse>` message pair is exchanged using the
411 HTTP POST-SimpleSign binding. The messages are signed as described in section 2.5, above. If the
412 messages were unsigned, they would be the same as shown below, except that the hidden form controls
413 named `Signature` and `SigAlg` would be missing.

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

```
415 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
416 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
417 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
418 21T19:00:49Z" Version="2.0">  
419 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
420 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
421 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
422 <samlp:SessionIndex>1</samlp:SessionIndex>  
423 </samlp:LogoutRequest>
```

424
425
426
427
428
429
430
431
432
433
434
435
436

```
<samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
  ID="b0730d21b628110d8b7e004005b13a2b"
  InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"
  IssueInstant="2004-01-21T19:00:49Z" Version="2.0">
  <Issuer>https://ServiceProvider.com/SAML</Issuer>
  <samlp:Status>
    <samlp:StatusCode
  Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
  </samlp:Status>
</samlp:LogoutResponse>
```

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

440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484

```
HTTP/1.1 200 OK
Date: 21 Jan 2004 07:00:49 GMT
Content-Type: text/html; charset=iso-8859-1

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<body onload="document.forms[0].submit()">

<noscript>
<p>
<strong>Note:</strong> Since your browser does not support JavaScript,
you must press the Continue button once to proceed.
</p>
</noscript>

<form action="http://ServiceProvider.com/SAML/SLO/Browser" method="post">
<div>
<input type="hidden" name="RelayState"
value="0043bfclbc45110dae17004005b13a2b"/>
<input type="hidden" name="SAMLRequest"
value="PHNhbWxwOmxvZ291dFJlcXVlc3QgeG1sbnM6c2FtbHA9InVybjpvYXNpczpuYW11
czp0YzptQU1MOjIuMDpwcm90b2NvbCIgeG1sbnM9InVybjpvYXNpczpuYW11czp0
YzptQU1MOjIuMDphc3NlcnRpb24iCiAgICBJRD0iZDZiN2MzODhjZWZmNmZhN2Mz
OWMyOGZkMjk4NjQ0YTgiIElzc3VlSW5zdGFudD0iMjAwNC0wMS0yMVQxOTowMDo0
OVoiIFZlcnNpb249IjIuMCI+CiAgICA8SXNzdWVpPmh0dHBzOi8vSWRlbnRpdHlQ
cm92aWRlci5jb20vU0FNTDdWvSXNzdWVpPggogICAgPE5hbWVJRCBGb3JtYXQ9InVy
bjpvYXNpczpuYW11czp0YzptQU1MOjIuMDpuYW1laWQtZm9ybWF0OnBlcnNpc3Rl
bnQiPjAwNWwWmUwLWFkODI0MTEwZC1hNTU2LTAwNDAwNW1xM2EyYjwvTmFtZU1E
PgogICAgPHNhbWxwO1NlcnRpb25JbmlleD4xPC9zYW1scDpTZXRzaW9uSW5kZXZg+
Cjwvc2FtbHA6TG9nb3V0UmVxdWVzdD4K"/>
<input type="hidden" name="Signature"
value="J4if7CCeHVfn4H6hMZN5fijOjQIyZ/laoFUZWz4LCRN3J82UeoyYvAiTD0QOUZHT
RJNU1lWgub1pw4QR9MH5bwfLEa8XDivA118dR0Q7YN5L/U5rmbxnglQ9pV0jT44c
RNeqtbLW0YF4p1fcqg7E5i0SljE3QLkiaAdkAec2a4HwPFkn/JP7w011Mc6kU8ML
CBbZAa3+94ZvVwHBEdyCdU+lyEvf+JGxTww66BwI2ugmPFxvoJdsOOAWwS3KhAFhL
LSPXnhb3nd/ovKNNV/khZYwqsFTFNTMA+0JraKsZiCrTEZzEPXaP9KilrjPIIvRV
xDQhETj96flk5zmkEM3ruw=="/>
<input type="hidden" name="SigAlg"
value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
</div>
<noscript>
<div>
<input type="submit" value="Continue"/>
```

```
485 </div>
486 </noscript>
487 </form>
488 </body>
489 </html>
490
```

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

```
494 HTTP/1.1 200 OK
495 Date: 21 Jan 2004 07:00:49 GMT
496 Content-Type: text/html; charset=iso-8859-1
497
498 <?xml version="1.0" encoding="UTF-8"?>
499 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
500 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
501 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
502 <body onload="document.forms[0].submit()">
503
504 <noscript>
505 <p>
506 <strong>Note:</strong> Since your browser does not support JavaScript,
507 you must press the Continue button once to proceed.
508 </p>
509 </noscript>
510
511 <form action="https://IdentityProvider.com/SAML/SLO/Response"
512 method="post">
513 <div>
514 <input type="hidden" name="RelayState"
515 value="0043bfc1bc45110dae17004005b13a2b"/>
516 <input type="hidden" name="SAMLResponse"
517 value="PHNhbWxwOkxvZ291dFJlcXVlc3QgeG1sbnM6c2FtbHA9InVybjpvYXNpczpuYW11
518 czp0YzptQU1MOjIuMDpwcm90b2NvbCIgeG1sbnM9InVybjpvYXNpczpuYW11czp0
519 YzptQU1MOjIuMDphc3NlcnRpb24iCiAgICBjRD0iZDZiN2MzODhjZWZmNmZhN2Mz
520 OWMyOGZkMjk4NjQ0YTgiIElzc3VlSW5zdGFudD0iMjAwNC0yMVQxOTowMDo0
521 OVoiIFZlcnNpb249IjIuMCI+CiAgICAgICA8SXNzdWVYpGogICAgPE5hbWVJRCBGb3JtYXQ9InVy
522 cm92aWRlci5jb20vU0FNTDdwSXNzdWVYpGogICAgPE5hbWVJRCBGb3JtYXQ9InVy
523 bjpvYXNpczpuYW11czp0YzptQU1MOjIuMDpuYW11aWQtZm9ybWF0OnBlcnNpc3Rl
524 bnQiPjAwNWUwNmUwLWFkODItMTEwZC1hNTU2LTAwNDAwNW1xM2EyYjwvTmFtZU1E
525 PgogICAgPHNhbWxwOlNlc3Npb25JbmlleD4xPC9zYW1scDpTZXRzaW9uSW5kZXg+
526 Cjwvc2FtbHA6TG9nb3V0UmVxdWVzdD4K"/>
527 <input type="hidden" name="Signature"
528 value="DCDqAwIdQSwyXGvG2YvNjnj7Plkt0+kbCfRj9gGTrN4KKPqvQl5EsFrWRkMOdx
529 xuwPldWPKvfgX6rt+pKwLgCt1TqRj+71y+VdGS8ORsBeEIURRn9wSu+pKsWiHexw
530 KnIe65bjONbg2db44QOWZLDe76fLi05Psy/7HZTQuModRFYSR//VyNGHQmf9Sxi6
531 mkmrYMXPOyZAUfNhX4eLaXffwCHT0yRrEcm/PAEDDa7uqe8Uo5ilstgXDWDodWdk
532 Szk8ZS1irjFkvtxH7FJlm9ADt1W/SoX92jGjMIrdQwCyArI6o8KtiDp/cjDjHZGi
533 XLx2WvS7GEibA7Qd+5hSBQ==" />
534 <input type="hidden" name="SigAlg"
535 value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
536 </div>
537 <noscript>
538 <div>
539 <input type="submit" value="Continue"/>
540 </div>
541 </noscript>
542 </form>
543 </body>
544 </html>
```

545 **Appendix A. Acknowledgments**

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

548

- 549 • Hal Lockhart, BEA Systems, Inc.
- 550 • Rob Philpott, EMC Corporation
- 551 • Scott Cantor, Internet2
- 552 • Bob Morgan, Internet2
- 553 • Eric Tiffany, Liberty Alliance Project
- 554 • Tom Scavo, National Center for Supercomputing Applications (NCSA)
- 555 • Peter Davis, Neustar, Inc.
- 556 • Jeff Hodges, Neustar, Inc.
- 557 • Frederick Hirsch, Nokia Corporation
- 558 • Abbie Barbir, Nortel Networks Limited
- 559 • Paul Madsen, NTT Corporation
- 560 • Ari Kermaier, Oracle Corporation
- 561 • Prateek Mishra, Oracle Corporation
- 562 • Brian Campbell, Ping Identity Corporation
- 563 • Anil Saldhana, Red Hat
- 564 • Eve Maler, Sun Microsystems
- 565 • Emily Xu, Sun Microsystems
- 566 • Kent Spaulding, Tripod Technology Group, Inc.
- 567 • David Staggs, Veterans Health Administration
- 568