



1 Bindings for the OASIS Security 2 Assertion Markup Language (SAML) 3 V2.0 4

5 **Committee Draft 02, 24 September 2004**

6 **Document identifier:**

7 sstc-saml-bindings-2.0-cd-02

8 **Location:**

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

10 **Editors:**

11 Scott Cantor, Internet2
12 Frederick Hirsch, Nokia
13 John Kemp, Nokia
14 Rob Philpott, RSA Security
15 Eve Maler, Sun Microsystems

16 **SAML V2.0 Contributors:**

17 Conor P. Cahill, AOL
18 Hal Lockhart, BEA Systems
19 Michael Beach, Boeing
20 Rick Randall, Booze, Allen, Hamilton
21 Tim Alsop, CyberSafe Limited
22 Nick Ragouzis, Enosis
23 John Hughes, Atos Origin
24 Paul Madsen, Entrust
25 Irving Reid, Hewlett-Packard
26 Paula Austel, IBM
27 Maryann Hondo, IBM
28 Michael McIntosh, IBM
29 Tony Nadalin, IBM
30 Scott Cantor, Internet2
31 RL 'Bob' Morgan, Internet2
32 Rebekah Metz, NASA
33 Prateek Mishra, Netegrity
34 Peter C Davis, Neustar
35 Frederick Hirsch, Nokia
36 John Kemp, Nokia
37 Charles Knouse, Oblix
38 Steve Anderson, OpenNetwork
39 John Linn, RSA Security
40 Rob Philpott, RSA Security
41 Jahan Moreh, Sigaba
42 Anne Anderson, Sun Microsystems
43 Jeff Hodges, Sun Microsystems
44 Eve Maler, Sun Microsystems

45 Ron Monzillo, Sun Microsystems
46 Greg Whitehead, Trustgenix

47 **Abstract:**

48 This specification defines protocol bindings for the use of SAML assertions and request-response
49 messages in communications protocols and frameworks.

50 **Status:**

51 This is a **second Committee Draft** approved by the Security Services Technical Committee on
52 21 September 2004.

53 Committee members should submit comments and potential errata to the [security-](mailto:security-services@lists.oasis-open.org)
54 [services@lists.oasis-open.org](mailto:security-services@lists.oasis-open.org) list. Others should submit them by filling out the web form located
55 at http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=security. The
56 committee will publish on its web page (<http://www.oasis-open.org/committees/security>) a catalog
57 of any changes made to this document as a result of comments.

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

62 Table of Contents

63	1 Introduction.....	5
64	1.1 Protocol Binding Concepts.....	5
65	1.2 Notation.....	5
66	2 Guidelines for Specifying Additional Protocol Bindings.....	7
67	3 Protocol Bindings.....	8
68	3.1 General Considerations.....	8
69	3.1.1 Use of RelayState.....	8
70	3.1.2 Security.....	8
71	3.1.2.1 Use of SSL 3.0 or TLS 1.0.....	8
72	3.1.2.2 Data Origin Authentication.....	8
73	3.1.2.3 Message Integrity.....	8
74	3.1.2.4 Message Confidentiality.....	9
75	3.1.2.5 Security Considerations.....	9
76	3.2 SAML SOAP Binding.....	9
77	3.2.1 Required Information.....	9
78	3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding.....	10
79	3.2.2.1 Basic Operation.....	10
80	3.2.2.2 SOAP Headers.....	10
81	3.2.3 Use of SOAP over HTTP.....	11
82	3.2.3.1 HTTP Headers.....	11
83	3.2.3.2 Caching.....	11
84	3.2.3.3 Error Reporting.....	11
85	3.2.3.4 Metadata Considerations.....	12
86	3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP.....	12
87	3.3 Reverse SOAP (PAOS) Binding.....	13
88	3.3.1 Required Information.....	13
89	3.3.2 Overview.....	13
90	3.3.3 Message Exchange.....	13
91	3.3.3.1 HTTP Request, SAML Request in SOAP Response.....	14
92	3.3.3.2 SAML Response in SOAP Request, HTTP Response.....	15
93	3.3.4 Caching.....	15
94	3.3.5 Security Considerations.....	15
95	3.3.5.1 Error Reporting.....	15
96	3.3.5.2 Metadata Considerations.....	15
97	3.4 HTTP Redirect Binding.....	15
98	3.4.1 Required Information.....	16
99	3.4.2 Overview.....	16
100	3.4.3 RelayState.....	16
101	3.4.4 Message Encoding.....	16
102	3.4.4.1 DEFLATE Encoding.....	17
103	3.4.5 Message Exchange.....	18
104	3.4.5.1 HTTP and Caching Considerations.....	19
105	3.4.5.2 Security Considerations.....	19
106	3.4.6 Error Reporting.....	19
107	3.4.7 Metadata Considerations.....	20
108	3.4.8 Example SAML Message Exchange Using HTTP Redirect.....	20

109	3.5 HTTP POST Binding.....	21
110	3.5.1 Required Information.....	21
111	3.5.2 Overview.....	21
112	3.5.3 RelayState.....	21
113	3.5.4 Message Encoding.....	22
114	3.5.5 Message Exchange.....	22
115	3.5.5.1 HTTP and Caching Considerations.....	23
116	3.5.5.2 Security Considerations.....	24
117	3.5.6 Error Reporting.....	24
118	3.5.7 Metadata Considerations.....	24
119	3.5.8 Example SAML Message Exchange Using HTTP POST.....	24
120	3.6 HTTP Artifact Binding.....	26
121	3.6.1 Required Information.....	26
122	3.6.2 Overview.....	27
123	3.6.3 Message Encoding.....	27
124	3.6.3.1 RelayState.....	27
125	3.6.3.2 URL Encoding.....	27
126	3.6.3.3 Form Encoding.....	27
127	3.6.4 Artifact Format.....	28
128	3.6.4.1 Required Information.....	28
129	3.6.4.2 Format Details.....	28
130	3.6.5 Message Exchange.....	29
131	3.6.5.1 HTTP and Caching Considerations.....	31
132	3.6.5.2 Security Considerations.....	31
133	3.6.6 Error Reporting.....	32
134	3.6.7 Metadata Considerations.....	32
135	3.6.8 Example SAML Message Exchange Using HTTP Artifact.....	32
136	3.7 SAML URI Binding.....	35
137	3.7.1 Required Information.....	35
138	3.7.2 Protocol-Independent Aspects of the SAML URI Binding.....	35
139	3.7.2.1 Basic Operation.....	35
140	3.7.3 Security Considerations.....	36
141	3.7.4 MIME Encapsulation.....	36
142	3.7.5 Use of HTTP URIs.....	36
143	3.7.5.1 URI Syntax.....	36
144	3.7.5.2 HTTP and Caching Considerations.....	36
145	3.7.5.3 Security Considerations.....	36
146	3.7.5.4 Error Reporting.....	37
147	3.7.5.5 Metadata Considerations.....	37
148	3.7.5.6 Example SAML Message Exchange Using an HTTP URI.....	37
149	4 References.....	38
150	5 Registration of MIME media type application/samlassertion+xml.....	40

1 Introduction

151

152 This document specifies SAML protocol bindings for the use of SAML assertions and request-response
153 messages in communications protocols and frameworks.

154 [SAMLCore] defines the SAML assertions and request-response messages themselves, and
155 [SAMLProfile] defines specific usage patterns that reference both [SAMLCore] and bindings defined in this
156 specification or elsewhere.

1.1 Protocol Binding Concepts

157

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

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

164 The intent of this specification is to specify a selected set of bindings in sufficient detail to ensure that
165 independently implemented SAML-conforming software can interoperate when using standard messaging
166 or communication protocols.

167 Unless otherwise specified, a binding should be understood to support the transmission of any SAML
168 protocol message derived from the **samlp:RequestAbstractType** and **samlp:StatusResponseType**
169 types. Further, when a binding refers to "SAML requests and responses", it should be understood to mean
170 any protocol messages derived from those types.

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

1.2 Notation

172

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

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

177 `Example code listings appear like this.`

178 **Note:** Non-normative notes and explanations appear like this.

179 Conventional XML namespace prefixes are used throughout this specification to stand for their respective
180 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].
ds:	http://www.w3.org/2000/09/xmldsig#	This namespace is defined in the XML Signature Syntax and Processing specification [XMLSig] and its governing schema.

Prefix	XML Namespace	Comments
SOAP-ENV:	http://schemas.xmlsoap.org/soap/envelope	This namespace is defined in SOAP V1.1 [SOAP1.1].

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

2 Guidelines for Specifying Additional Protocol Bindings

184
185

186 This specification defines a selected set of protocol bindings, but others will possibly be developed in the
187 future. It is not possible for the OASIS Security Services Technical Committee (SSTC) to standardize all of
188 these additional bindings for two reasons: it has limited resources and it does not own the standardization
189 process for all of the technologies used. This section offers guidelines for third parties who wish to specify
190 additional bindings.

191 The SSTC welcomes submission of proposals from OASIS members for new protocol bindings. OASIS
192 members may wish to submit these proposals for consideration by the SSTC in a future version of this
193 specification. Other members may simply wish to inform the committee of their work related to SAML.
194 Please refer to the SSTC web site for further details on how to submit such proposals to the SSTC.

195 Following is a checklist of issues that **MUST** be addressed by each protocol binding:

- 196 1. Specify three pieces of identifying information: a URI that uniquely identifies the protocol binding,
197 postal or electronic contact information for the author, and a reference to previously defined
198 bindings or profiles that the new binding updates or obsoletes.
- 199 2. Describe the set of interactions between parties involved in the binding. Any restrictions on
200 applications used by each party and the protocols involved in each interaction must be explicitly
201 called out.
- 202 3. Identify the parties involved in each interaction, including how many parties are involved and
203 whether intermediaries may be involved.
- 204 4. Specify the method of authentication of parties involved in each interaction, including whether
205 authentication is required and acceptable authentication types.
- 206 5. Identify the level of support for message integrity, including the mechanisms used to ensure
207 message integrity.
- 208 6. Identify the level of support for confidentiality, including whether a third party may view the contents
209 of SAML messages and assertions, whether the binding requires confidentiality, and the
210 mechanisms recommended for achieving confidentiality.
- 211 7. Identify the error states, including the error states at each participant, especially those that receive
212 and process SAML assertions or messages.
- 213 8. Identify security considerations, including analysis of threats and description of countermeasures.
- 214 9. Identify metadata considerations, such that support for a binding involving a particular
215 communications protocol or used in a particular profile can be advertised in an efficient and
216 interoperable way.

217 **3 Protocol Bindings**

218 The following sections define the protocol bindings that are specified as part of the SAML standard.

219 **3.1 General Considerations**

220 The following sections describe normative characteristics of all protocol bindings defined for SAML.

221 **3.1.1 Use of RelayState**

222 Some bindings define a "RelayState" mechanism for preserving and conveying state information. When
223 such a mechanism is used in conveying a request message as the initial step of a SAML protocol, it
224 places requirements on the selection and use of the binding subsequently used to convey the response.
225 Namely, if a SAML request message is accompanied by RelayState data, then the SAML responder
226 MUST return its SAML protocol response using a binding that also supports a RelayState mechanism, and
227 it MUST place the exact RelayState data it received with the request into the corresponding RelayState
228 parameter in the response.

229 **3.1.2 Security**

230 Unless stated otherwise, these security statements about apply to all bindings. Bindings may also make
231 additional statements about these security features.

232 **3.1.2.1 Use of SSL 3.0 or TLS 1.0**

233 Unless otherwise specified, in any SAML binding's use of SSL 3.0 [SSL3] or TLS 1.0 [RFC2246], servers
234 MUST authenticate to clients using a X.509 v3 certificate. The client MUST establish server identity based
235 on contents of the certificate (typically through examination of the certificate's subject DN field,
236 subjectAltName attribute, etc.).

237 **3.1.2.2 Data Origin Authentication**

238 Authentication of both the SAML requester and the SAML responder associated with a message is
239 OPTIONAL and depends on the environment of use. Authentication mechanisms available at the SOAP
240 message exchange layer or from the underlying substrate protocol (for example in many bindings the
241 SSL/TLS or HTTP protocol) MAY be utilized to provide data origin authentication.

242 Transport authentication will not meet end-end origin-authentication requirements in bindings where the
243 SAML protocol message passes through an intermediary – in this case message authentication is
244 recommended.

245 Note that SAML itself offers mechanisms for parties to authenticate to one another, but in addition SAML
246 may use other authentication mechanisms to provide security for SAML itself.

247 **3.1.2.3 Message Integrity**

248 Message integrity of both SAML requests and SAML responses is OPTIONAL and depends on the
249 environment of use. The security layer in the underlying substrate protocol or a mechanism at the SOAP
250 message exchange layer MAY be used to ensure message integrity.

251 Transport integrity will not meet end-end integrity requirements in bindings where the SAML protocol
252 message passes through an intermediary – in this case message integrity is recommended.

253 3.1.2.4 Message Confidentiality

254 Message confidentiality of both SAML requests and SAML responses is OPTIONAL and depends on the
255 environment of use. The security layer in the underlying substrate protocol or a mechanism at the SOAP
256 message exchange layer MAY be used to ensure message confidentiality.

257 Transport confidentiality will not meet end-end confidentiality requirements in bindings where the SAML
258 protocol message passes through an intermediary.

259 3.1.2.5 Security Considerations

260 Before deployment, each combination of authentication, message integrity, and confidentiality
261 mechanisms SHOULD be analyzed for vulnerability in the context of the specific protocol exchange and
262 the deployment environment. See specific protocol processing rules in [SAMLCore] and the SAML security
263 considerations document [SAMLSecure] for a detailed discussion.

264 [RFC2617] describes possible attacks in the HTTP environment when basic or message-digest
265 authentication schemes are used.

266 Special care should be given to the impact of possible caching on security.

267 3.2 SAML SOAP Binding

268 SOAP is a lightweight protocol intended for exchanging structured information in a decentralized,
269 distributed environment [SOAP1.1]. It uses XML technologies to define an extensible messaging
270 framework providing a message construct that can be exchanged over a variety of underlying protocols.
271 The framework has been designed to be independent of any particular programming model and other
272 implementation specific semantics. Two major design goals for SOAP are simplicity and extensibility.
273 SOAP attempts to meet these goals by omitting, from the messaging framework, features that are often
274 found in distributed systems. Such features include but are not limited to "reliability", "security",
275 "correlation", "routing", and "Message Exchange Patterns" (MEPs).

276 A SOAP message is fundamentally a one-way transmission between SOAP nodes from a SOAP sender
277 to a SOAP receiver, possibly routed through one or more SOAP intermediaries. SOAP messages are
278 expected to be combined by applications to implement more complex interaction patterns ranging from
279 request/response to multiple, back-and-forth "conversational" exchanges [SOAP-PRIMER].

280 SOAP defines an XML message envelope that includes header and body sections, allowing data and
281 control information to be transmitted. SOAP also defines processing rules associated with this envelope
282 and an HTTP binding for SOAP message transmission.

283 The SAML SOAP binding defines how to use SOAP to send and receive SAML requests and responses.

284 Like SAML, SOAP can be used over multiple underlying transports. This binding has protocol-independent
285 aspects, but also calls out the use of SOAP over HTTP as REQUIRED (mandatory to implement).

286 3.2.1 Required Information

287 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:SOAP

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

289 **Description:** Given below.

290 **Updates:** urn:oasis:names:tc:SAML:1.0:bindings:SOAP-binding

291 3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding

292 The following sections define aspects of the SAML SOAP binding that are independent of the underlying
293 protocol, such as HTTP, on which the SOAP messages are transported. Note this binding only supports
294 the use of SOAP 1.1.

295 3.2.2.1 Basic Operation

296 SOAP 1.1 messages consist of three elements: an envelope, header data, and a message body. SAML
297 request-response protocol elements MUST be enclosed within the SOAP message body.

298 SOAP 1.1 also defines an optional data encoding system. This system is not used within the SAML SOAP
299 binding. This means that SAML messages can be transported using SOAP without re-encoding from the
300 "standard" SAML schema to one based on the SOAP encoding.

301 The system model used for SAML conversations over SOAP is a simple request-response model.

- 302 1. A system entity acting as a SAML requester transmits a SAML request element within the body of
303 a SOAP message to a system entity acting as a SAML responder. The SAML requester MUST
304 NOT include more than one SAML request per SOAP message or include any additional XML
305 elements in the SOAP body.
- 306 2. The SAML responder MUST return either a SAML response element within the body of another
307 SOAP message or generate a SOAP fault. The SAML responder MUST NOT include more than
308 one SAML response per SOAP message or include any additional XML elements in the SOAP
309 body. If a SAML responder cannot, for some reason, process a SAML request, it MUST generate a
310 SOAP fault. SOAP fault codes MUST NOT be sent for errors within the SAML problem domain, for
311 example, inability to find an extension schema or as a signal that the subject is not authorized to
312 access a resource in an authorization query. (SOAP 1.1 faults and fault codes are discussed in
313 [SOAP1.1] §4.1.)

314 On receiving a SAML response in a SOAP message, the SAML requester MUST NOT send a fault code
315 or other error messages to the SAML responder. Since the format for the message interchange is a
316 simple request-response pattern, adding additional items such as error conditions would needlessly
317 complicate the protocol.

318 [SOAP1.1] references an early draft of the XML Schema specification including an obsolete namespace.
319 SAML requesters SHOULD generate SOAP documents referencing only the final XML schema
320 namespace. SAML responders MUST be able to process both the XML schema namespace used in
321 [SOAP1.1] as well as the final XML schema namespace.

322 3.2.2.2 SOAP Headers

323 A SAML requester in a SAML conversation over SOAP MAY add arbitrary headers to the SOAP message.
324 This binding does not define any additional SOAP headers.

325 **Note:** The reason other headers need to be allowed is that some SOAP software and
326 libraries might add headers to a SOAP message that are out of the control of the SAML-
327 aware process. Also, some headers might be needed for underlying protocols that require
328 routing of messages or by message security mechanisms.

329 A SAML responder MUST NOT require any headers in the SOAP message in order to process the SAML
330 message correctly itself, but MAY require additional headers that address underlying routing or message
331 security requirements.

332 **Note:** The rationale is that requiring extra headers will cause fragmentation of the SAML
333 standard and will hurt interoperability.

334 3.2.3 Use of SOAP over HTTP

335 A SAML processor that claims conformance to the SAML SOAP binding MUST implement SAML over
336 SOAP over HTTP. This section describes certain specifics of using SOAP over HTTP, including HTTP
337 headers, caching, and error reporting.

338 The HTTP binding for SOAP is described in [SOAP1.1] §6.0. It requires the use of a `SOAPAction` header
339 as part of a SOAP HTTP request. A SAML responder MUST NOT depend on the value of this header. A
340 SAML requester MAY set the value of `SOAPAction` header as follows:

341 `http://www.oasis-open.org/committees/security`

342 3.2.3.1 HTTP Headers

343 A SAML requester in a SAML conversation over SOAP over HTTP MAY add arbitrary headers to the
344 HTTP request. This binding does not define any additional HTTP headers.

345 **Note:** The reason other headers need to be allowed is that some HTTP software and
346 libraries might add headers to an HTTP message that are out of the control of the SAML-
347 aware process. Also, some headers might be needed for underlying protocols that require
348 routing of messages or by message security mechanisms.

349 A SAML responder MUST NOT require any headers in the HTTP request to correctly process the SAML
350 message itself, but MAY require additional headers that address underlying routing or message security
351 requirements.

352 **Note:** The rationale is that requiring extra headers will cause fragmentation of the SAML
353 standard and will hurt interoperability.

354 3.2.3.2 Caching

355 HTTP proxies should not cache SAML protocol messages. To insure this, the following rules SHOULD be
356 followed.

357 When using HTTP 1.1, requesters SHOULD:

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

360 When using HTTP 1.1, responders SHOULD:

- 361 • Include a `Cache-Control` header field set to "no-cache, no-store, must-revalidate,
362 private".
- 363 • Include a `Pragma` header field set to "no-cache".
- 364 • NOT include a `Validator`, such as a `Last-Modified` or `ETag` header.

365 3.2.3.3 Error Reporting

366 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD
367 return a "403 Forbidden" response. In this case, the content of the HTTP body is not significant.

368 As described in [SOAP1.1] § 6.2, in the case of a SOAP error while processing a SOAP request, the
369 SOAP HTTP server MUST return a "500 Internal Server Error" response and include a SOAP
370 message in the response with a SOAP <SOAP-ENV:fault> element. This type of error SHOULD be
371 returned for SOAP-related errors detected before control is passed to the SAML processor, or when the
372 SOAP processor reports an internal error (for example, the SOAP XML namespace is incorrect, the SAML
373 schema cannot be located, the SAML processor throws an exception, and so on).

374 In the case of a SAML processing error, the SOAP HTTP server MUST respond with "200 OK" and
375 include a SAML-specified <sampl:Status> element in the SAML response within the SOAP body. Note
376 that the <sampl:Status> element does not appear by itself in the SOAP body, but only within a SAML
377 response of some sort.

378 For more information about the use of SAML status codes, see the SAML assertions and protocols
379 specification [SAMLCore].

380 3.2.3.4 Metadata Considerations

381 Support for the SOAP binding SHOULD be reflected by indicating either a URL endpoint at which requests
382 contained in SOAP messages for a particular protocol or profile are to be sent, or alternatively with a
383 WSDL port/endpoint definition.

384 3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP

385 Following is an example of a query that asks for an assertion containing an attribute statement from a
386 SAML attribute authority.

```
387 POST /SamlService HTTP/1.1
388 Host: www.example.com
389 Content-Type: text/xml
390 Content-Length: nnn
391 SOAPAction: http://www.oasis-open.org/committees/security
392 <SOAP-ENV:Envelope
393   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
394   <SOAP-ENV:Body>
395     <sampl:AttributeQuery xmlns:sampl="..."
396   xmlns:saml="..." xmlns:ds="..." ID="_6c3a4f8b9c2d" Version="2.0"
397   IssueInstant="2004-03-27T08:41:00Z"
398     <ds:Signature> ... </ds:Signature>
399     <saml:Subject>
400       ...
401     </saml:Subject>
402   </sampl:AttributeQuery>
403 </SOAP-ENV:Body>
404 </SOAP-ENV:Envelope>
```

405 Following is an example of the corresponding response, which supplies an assertion containing the
406 attribute statement as requested.

```
407 HTTP/1.1 200 OK
408 Content-Type: text/xml
409 Content-Length: nnnn
410 <SOAP-ENV:Envelope
411   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
412   <SOAP-ENV:Body>
413     <sampl:Response xmlns:sampl="..." xmlns:saml="..." xmlns:ds="..."
414   ID="_6c3a4f8b9c2d" Version="2.0" IssueInstant="2004-03-27T08:42:00Z">
415       <saml:Issuer>https://www.example.com/SAML</saml:Issuer>
416       <ds:Signature> ... </ds:Signature>
417       <Status>
418         <StatusCode Value="..." />
419       </Status>
420
421       <saml:Assertion>
```

```
422         <saml:Subject>
423         ...
424         </saml:Subject>
425         <saml:AttributeStatement>
426         ...
427         </saml:AttributeStatement>
428         </saml:Assertion>
429     </samlp:Response>
430 </SOAP-Env:Body>
431 </SOAP-ENV:Envelope>
```

432 3.3 Reverse SOAP (PAOS) Binding

433 This binding leverages the Reverse HTTP Binding for SOAP specification [PAOS]. Implementers MUST
434 comply with the general processing rules specified in [PAOS] in addition to those specified in this
435 document. In case of conflict, [PAOS] is normative.

436 3.3.1 Required Information

437 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:PAOS

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

439 **Description:** Given below.

440 **Updates:** None.

441 3.3.2 Overview

442 The reverse SOAP binding is a mechanism by which an HTTP requester can advertise the ability to act as
443 a SOAP responder or a SOAP intermediary to a SAML requester. The HTTP requester is able to support
444 a pattern where a SAML request is sent to it in a SOAP envelope in an HTTP response from the SAML
445 requester, and the HTTP requester responds with a SAML response in a SOAP envelope in a subsequent
446 HTTP request. This message exchange pattern supports the use case defined in the ECP SSO profile
447 (described in the SAML profiles specification [SAMLProfile]), in which the HTTP requester is an
448 intermediary in an authentication exchange.

449 3.3.3 Message Exchange

450 The PAOS binding includes two component message exchange patterns:

- 451 1. The HTTP requester sends an HTTP request to a SAML requester. The SAML requester responds
452 with an HTTP response containing a SOAP envelope containing a SAML request message.
- 453 2. Subsequently, the HTTP requester sends an HTTP request to the original SAML requester
454 containing a SOAP envelope containing a SAML response message. The SAML requester
455 responds with an HTTP response, possibly in response to the original service request in step 1.

456 The ECP profile uses the PAOS binding to provide authentication of the client to the service provider
457 before the service is provided. This occurs in the following steps, illustrated in Figure A:

- 458 1. Client requests service using HTTP request.
- 459 2. Service Provider responds with a SAML authentication request. This is sent using a SOAP request,
460 carried in the HTTP response.
- 461 3. The Client returns a SOAP response carrying a SAML authentication response. This is sent using a
462 new HTTP request.
- 463 4. Assuming service provider authentication and authorization is successful the service provider may
464 respond to the original service request in the HTTP response.

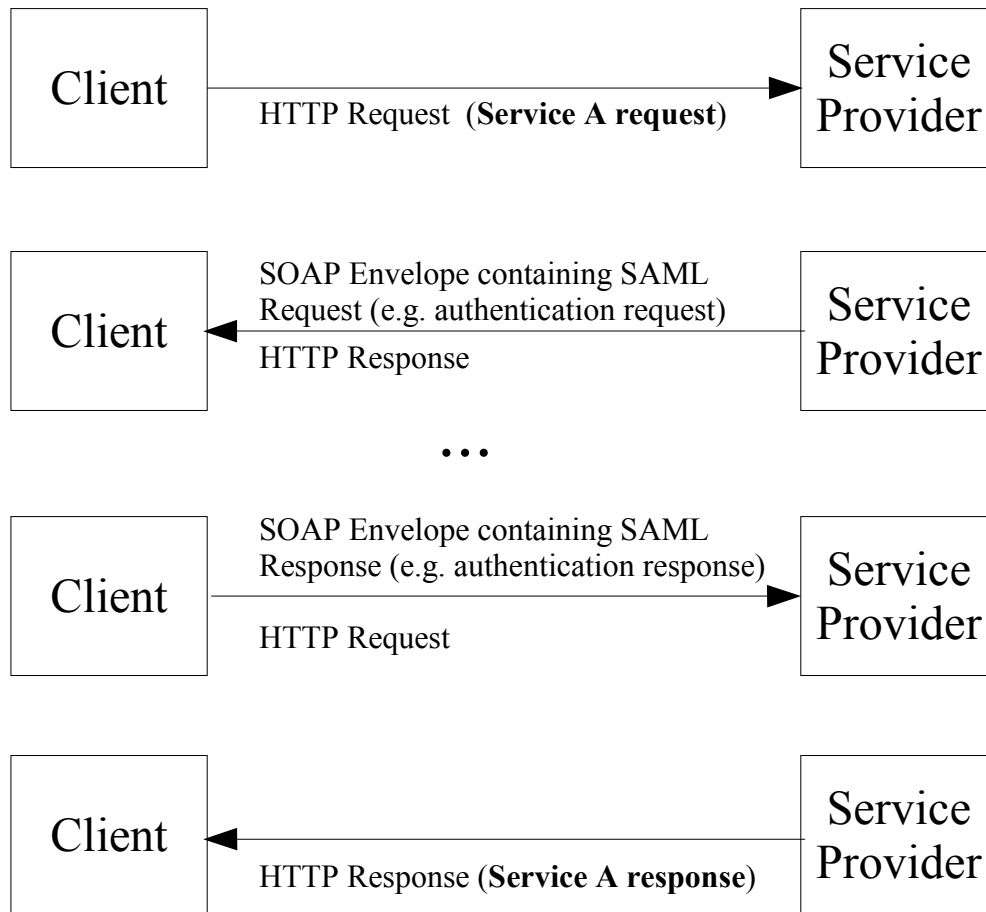


Figure 1: PAOS Binding Message Exchanges

465 The HTTP requester advertises the ability to handle this reverse SOAP binding in its HTTP requests using
 466 the HTTP headers defined by the PAOS specification. Specifically:

- 467 • The HTTP `Accept` Header field MUST indicate an ability to accept the
- 468 “application/vnd.paos+xml” content type.
- 469 • The HTTP `PAOS` Header field MUST be present and specify the PAOS version with
- 470 “urn:liberty:paos:2003-08” at a minimum.

471 Additional PAOS headers such as the service value MAY be specified by profiles that use the PAOS
 472 binding. The HTTP requester MAY add arbitrary headers to the HTTP request.

473 Note that this binding does not define a RelayState mechanism. Specific profiles that make use of this
 474 binding must therefore define such a mechanism, if needed. The use of a SOAP header is suggested for
 475 this purpose.

476 The following sections provide more detail on the two steps of the message exchange.

477 3.3.3.1 HTTP Request, SAML Request in SOAP Response

478 In response to an arbitrary HTTP request, the HTTP responder MAY return a SAML request message
 479 using this binding by returning a SOAP 1.1 envelope in the HTTP response containing a single SAML
 480 request message in the SOAP body, with no additional body content. The SOAP envelope MAY contain
 481 arbitrary SOAP headers defined by PAOS, SAML profiles, or additional specifications.

482 Note that while the SAML request message is delivered to the HTTP requester, the actual intended
483 recipient MAY be another system entity, with the HTTP requester acting as an intermediary, as defined by
484 specific profiles.

485 **3.3.3.2 SAML Response in SOAP Request, HTTP Response**

486 When the HTTP requester delivers a SAML response message to the intended recipient using the PAOS
487 binding, it places it as the only element in the SOAP body in a SOAP envelope in an HTTP request. The
488 HTTP requester may or may not be the originator of the SAML response. The SOAP envelope MAY
489 contain arbitrary SOAP headers defined by PAOS, SAML profiles, or additional specifications. The SAML
490 exchange is considered complete and the HTTP response is unspecified by this binding.

491 Profiles MAY define additional constraints on the HTTP content of non-SOAP responses during the
492 exchanges covered by this binding.

493 **3.3.4 Caching**

494 HTTP proxies should not cache SAML protocol messages. To insure this, the following rules SHOULD be
495 followed.

496 When using HTTP 1.1, requesters sending SAML protocol messages SHOULD:

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

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

- 500 • Include a `Cache-Control` header field set to "no-cache, no-store, must-revalidate,
501 private".
- 502 • Include a `Pragma` header field set to "no-cache".
- 503 • NOT include a Validator, such as a `Last-Modified` or `ETag` header.

504 **3.3.5 Security Considerations**

505 The HTTP requester in the PAOS binding may act as a SOAP intermediary and when it does, transport
506 layer security for origin authentication, integrity and confidentiality may not meet end-end security
507 requirements. In this case security at the SOAP message layer is recommended.

508 **3.3.5.1 Error Reporting**

509 Standard HTTP and SOAP error conventions MUST be observed. Errors that occur during SAML
510 processing MUST NOT be signaled at the HTTP or SOAP layer and MUST be handled using SAML
511 response messages with an error `<samlp:Status>` element.

512 **3.3.5.2 Metadata Considerations**

513 Support for the PAOS binding SHOULD be reflected by indicating a URL endpoint at which HTTP
514 requests and/or SAML protocol messages contained in SOAP envelopes for a particular protocol or profile
515 are to be sent. Either a single endpoint or distinct request and response endpoints MAY be supplied.

516 **3.4 HTTP Redirect Binding**

517 The HTTP Redirect binding defines a mechanism by which SAML protocol messages can be transmitted
518 within URL parameters. Permissible URL length is theoretically infinite, but unpredictably limited in

519 practice. Therefore, specialized encodings are needed to carry XML messages on a URL, and larger or
520 more complex message content can be sent using the HTTP POST or Artifact bindings.

521 This binding MAY be composed with the HTTP POST binding (see Section 3.5) and the HTTP Artifact
522 binding (see Section 3.6) to transmit request and response messages in a single protocol exchange using
523 two different bindings.

524 This binding involves the use of a message encoding. While the definition of this binding includes the
525 definition of one particular message encoding, others MAY be defined and used.

526 **3.4.1 Required Information**

527 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect

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

529 **Description:** Given below.

530 **Updates:** None.

531 **3.4.2 Overview**

532 The HTTP Redirect binding is intended for cases in which the SAML requester and responder need to
533 communicate using an HTTP user agent (as defined in HTTP 1.1 [RFC2616]) as an intermediary. This
534 may be necessary, for example, if the communicating parties do not share a direct path of communication.
535 It may also be needed if the responder requires an interaction with the user agent in order to fulfill the
536 request, such as when the user agent must authenticate to it.

537 Note that some HTTP user agents may have the capacity to play a more active role in the protocol
538 exchange and may support other bindings that use HTTP, such as the SOAP and Reverse SOAP
539 bindings. This binding assumes nothing apart from the capabilities of a common web browser.

540 **3.4.3 RelayState**

541 RelayState data MAY be included with a SAML protocol message transmitted with this binding. The value
542 MUST NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the
543 message independent of any other protections that may or may not exist during message transmission.

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

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

551 **3.4.4 Message Encoding**

552 Messages are encoded for use with this binding using a URL encoding technique, and transmitted using
553 the HTTP GET method. There are many possible ways to encode XML into a URL, depending on the
554 constraints in effect. This specification defines one such method without precluding others. Binding
555 endpoints SHOULD indicate which encodings they support using metadata, when appropriate. Particular
556 encodings MUST be uniquely identified with a URI when defined. It is not a requirement that all possible
557 SAML messages be encodable with a particular set of rules, but the rules MUST clearly indicate which
558 messages or content can or cannot be so encoded.

559 A URL encoding MUST place the message entirely within the URL query string, and MUST reserve the
560 rest of the URL for the endpoint of the message recipient.

561 A query string parameter named `SAMLEncoding` is reserved to identify the encoding mechanism used. If
562 this parameter is omitted, then the value is assumed to be
563 `urn:oasis:names:tc:SAML:2.0:bindings:URL-Encoding:DEFLATE`.

564 All endpoints that support this binding MUST support the DEFLATE encoding described in the following
565 sub-section.

566 3.4.4.1 DEFLATE Encoding

567 **Identification:** `urn:oasis:names:tc:SAML:2.0:bindings:URL-Encoding:DEFLATE`

568 SAML protocol messages can be encoded into a URL via the DEFLATE compression method (see
569 [RFC1951]). In such an encoding, the following procedure should be applied to the original SAML protocol
570 message's XML serialization:

- 571 1. Any signature on the SAML protocol message, including the `<ds:Signature>` XML element itself,
572 MUST be removed. Note that if the content of the message includes another signature, such as a
573 signed SAML assertion, this embedded signature is not removed. However, the length of such a
574 message after encoding essentially precludes using this mechanism. Thus SAML protocol
575 messages that contain signed content SHOULD NOT be encoded using this mechanism.
- 576 2. The DEFLATE compression mechanism, as specified in [RFC1951] is then applied to the entire
577 remaining XML content of the original SAML protocol message.
- 578 3. The compressed data is subsequently base64-encoded according to the rules specified in
579 [RFC2045]. Linefeeds or other whitespace MUST be removed from the result.
- 580 4. The base-64 encoded data is then URL-encoded, and added to the URL as a query string
581 parameter which MUST be named `SAMLRequest` (if the message is a SAML request) or
582 `SAMLResponse` (if the message is a SAML response).
- 583 5. If the original SAML protocol message was signed using an XML digital signature, a new signature
584 covering the encoded data as specified above MUST be attached using the rules stated below.
- 585 6. If RelayState data is to accompany the SAML protocol message, it MUST be URL-encoded and
586 placed in an additional query string parameter named `RelayState`.

587 XML digital signatures are not directly URL-encoded according to the above rules, due to space concerns.
588 If the underlying SAML protocol message is signed with an XML signature [XMLSig], the URL-encoded
589 form of the message MUST be signed as follows:

- 590 1. The signature algorithm identifier MUST be included as an additional query string parameter,
591 named `SigAlg`. The value of this parameter MUST be a URI that identifies the algorithm used to
592 sign the URL-encoded SAML protocol message, specified according to [XMLSig] or whatever
593 specification governs the algorithm.
- 594 2. To construct the signature, a string consisting of the concatenation of the `RelayState` (if present),
595 `SigAlg`, and `SAMLRequest` (or `SAMLResponse`) query string parameters is constructed in one of
596 the following ways:

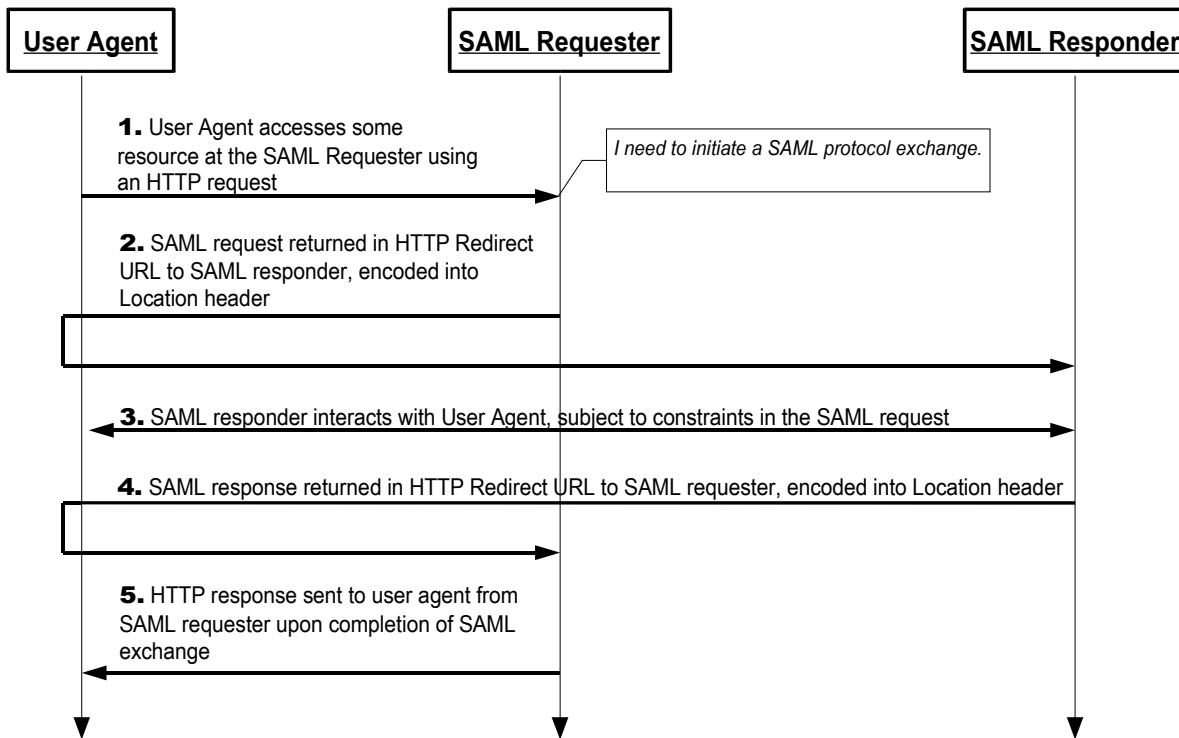
```
597 SAMLRequest=value&RelayState=value&SigAlg=value  
598 SAMLResponse=value&RelayState=value&SigAlg=value
```

- 599 3. The resulting string of bytes is the octet string to be fed into the signature algorithm. Any other
600 content in the original query string is not included and not signed.
- 601 4. The signature value MUST be encoded using the base64 encoding [RFC2045] with any whitespace
602 removed, and included as a query string parameter named `Signature`. Note that some characters
603 in the base64-encoded signature value may themselves require URL-encoding before being added.

- 604 5. The following signature algorithms (see [XMLSig]) and their URI representations MUST be
 605 supported with this encoding mechanism:
- 606 • DSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#dsa-sha1>
 - 607 • RSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#rsa-sha1>

608 3.4.5 Message Exchange

609 The system model used for SAML conversations via this binding is a request-response model, but these
 610 messages are sent to the user agent in an HTTP response and delivered to the message recipient in an
 611 HTTP request. The HTTP interactions before, between, and after these exchanges take place is
 612 unspecified. Both the SAML requester and the SAML responder are assumed to be HTTP responders.
 613 See the following sequence diagram illustrating the messages exchanged.



- 614 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
 615 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 616 2. The system entity acting as a SAML requester responds to the HTTP request from the user agent in
 617 step 1 by returning a SAML request. The SAML request is returned encoded into the HTTP
 618 response's Location header, and the HTTP status MUST be either 303 or 302. The SAML requester
 619 MAY include additional presentation and content in the HTTP response to facilitate the user agent's
 620 transmission of the message, as defined in HTTP 1.1 [RFC2616]. The user agent delivers the
 621 SAML request by issuing an HTTP GET request to the SAML responder.
- 622 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
 623 SAML response or MAY return arbitrary content to facilitate subsequent interaction with the user
 624 agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to
 625 indicate the requester's level of willingness to permit this kind of interaction (for example, the
 626 `IsPassive` attribute in `<samlp:AuthnRequest>`).

- 627 4. Eventually the responder SHOULD return a SAML response to the user agent to be returned to the
628 SAML requester. The SAML response is returned in the same fashion as described for the SAML
629 request in step 2.
- 630 5. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
631 user agent.

632 3.4.5.1 HTTP and Caching Considerations

633 HTTP proxies and the user agent intermediary should not cache SAML protocol messages. To insure this,
634 the following rules SHOULD be followed.

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

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

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

639 3.4.5.2 Security Considerations

640 The presence of the user agent intermediary means that the requester and responder cannot rely on the
641 transport layer for end-end authentication, integrity and confidentiality. URL-encoded messages MAY be
642 signed to provide origin authentication and integrity if the encoding method specifies a means for signing.

643 If the message is signed, the `Destination` XML attribute in the root SAML element of the protocol
644 message MUST contain the URL to which the sender has instructed the user agent to deliver the
645 message. The recipient MUST then verify that the value matches the location at which the message has
646 been received.

647 This binding SHOULD NOT be used if the content of the request or response should not be exposed to
648 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is
649 OPTIONAL and depends on the environment of use. If confidentiality is necessary, SSL 3.0 or TLS 1.0
650 SHOULD be used to protect the message in transit between the user agent and the SAML requester and
651 responder.

652 Note also that URL-encoded messages may be exposed in a variety of HTTP logs as well as the HTTP
653 "Referer" header.

654 Before deployment, each combination of authentication, message integrity, and confidentiality
655 mechanisms SHOULD be analyzed for vulnerability in the context of the specific protocol exchange, and
656 the deployment environment. See specific protocol processing rules in [SAMLCore], and the SAML
657 security considerations document [SAMLSecure] for a detailed discussion.

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

660 3.4.6 Error Reporting

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

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

666 For more information about SAML status codes, see the SAML assertions and protocols specification
667 [SAMLCore].

668 3.4.7 Metadata Considerations

669 Support for the HTTP Redirect binding SHOULD be reflected by indicating URL endpoints at which
670 requests and responses for a particular protocol or profile should be sent. Either a single endpoint or
671 distinct request and response endpoints MAY be supplied.

672 3.4.8 Example SAML Message Exchange Using HTTP Redirect

673 In this example, a <LogoutRequest> and <LogoutResponse> message pair are exchanged using the
674 HTTP Redirect binding.

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

```
676 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
677 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
678 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
679 21T19:00:49Z" Version="2.0">  
680 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
681 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
682 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
683 <samlp:SessionIndex>1</samlp:SessionIndex>  
684 </samlp:LogoutRequest>
```

```
685 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
686 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
687 ID="b0730d21b628110d8b7e004005b13a2b"  
688 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"  
689 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">  
690 <Issuer>https://ServiceProvider.com/SAML</Issuer>  
691 <samlp:Status>  
692 <samlp:StatusCode  
693 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>  
694 </samlp:Status>  
695 </samlp:LogoutResponse>
```

696 The initial HTTP request from the user agent in step 1 is not defined by this binding. To initiate the logout
697 protocol exchange, the SAML requester returns the following HTTP response, containing a signed SAML
698 request message. The SAMLRequest parameter value is actually derived from the request message
699 above. The signature portion is only illustrative and not the result of an actual computation. Note that the
700 line feeds in the HTTP Location header below are an artifact of the document, and there are no line
701 feeds in the actual header value.

```
702 HTTP/1.1 302 Object Moved  
703 Date: 21 Jan 2004 07:00:49 GMT  
704 Location:  
705 https://ServiceProvider.com/SAML/SLO/Browser?SAMLRequest=H4sIAOCuDUEAA32R  
706 UUvDMBSF3wf9DyXvWZOsq23oCsIQcPuIGz74liWZVtqk5qYy%2F73puoGCLE%2Fhu%2Bfecw%  
707 2B3BNG1Pd%2FYNzv4Z%  
708 2F05aPDxqWsN8HNlhQZnuBXQADei08C95Lv77YazOeG9s95K26Kp5bZYAGjnG2tQNIvDq9crp  
709 NjhTi7yXGq5yI4i%  
710 2FAvJ8qNiRZ61qchRXAMMujbghfErxAhJMaGY0T0tOCE8LV5RvBUf1r1oB2F40ATQmF%  
711 2BAoGpyLM%2FDXPXufQ88SWqljW%  
712 2F895OzX43Sbi5t14z71slFee17DGHqdfxgXSf87ZQjaRQ%  
713 2BnqW8H3cAH2xQRchSkEwTLFTOMKVEYbFcZjhECqUDXQh2KJPJ6mo8XWenYUxSG6VPFS2Tf2g  
714 0u%2BI%2Fpww8mv0ALfRRUQBAAA%  
715 3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F%  
716 2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-  
717 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE  
718 Content-Type: text/html; charset=iso-8859-1
```

719 After any unspecified interactions may have taken place, the SAML responder returns the HTTP response
720 below containing the signed SAML response message. Again, the SAMLResponse parameter value is
721 actually derived from the response message above. The signature portion is only illustrative and not the
722 result of an actual computation.

```
723 HTTP/1.1 302 Object Moved
724 Date: 21 Jan 2004 07:00:49 GMT
725 Location:
726 https://IdentityProvider.com/SAML/SLO/Response?SAMLResponse=H4sIAKO3DUEAA
727 31RTWvDMAy991cE39vYTtY6pimM7VJoYSylh94cR90yEitYTtnPX5a0sDKoTtLT09PXmkzbdH
728 qHH9iHd6AOHUH03Ta09JjKWe%
729 2BdRkM1aWdaIB2sLp730y0XXHceAlps2FTymGyIwIcaHztFg21fc1byVcIrKcqlVELwSpUr4D
730 z1%
731 2FKkUiZEli7buNtUBc1bJcmUTpSzYZHk2g59Zqc6VzNQyTY26KhP1sHUUjAs5k4PgnIu5FAeR
732 ac51mplYtDdf6I%2FgaZhn4AxA7f4AnG1GqfWo5TefIXSk47gAf6ktvHm81BX4hcU2%
733 2F1lwHV%2BJU9V01SKY0NME%2FYNfsILoaJoeH1%2BNRrYuemuBiMXXDvF9i1t8%
734 2F8jN7AcCxjwc4AEAAA%3D%
735 3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F%
736 2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-
737 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE
738 Content-Type: text/html; charset=iso-8859-1
```

739 3.5 HTTP POST Binding

740 The HTTP POST binding defines a mechanism by which SAML protocol messages may be transmitted
741 within the base64-encoded content of an HTML form control.

742 This binding MAY be composed with the HTTP Redirect binding (see Section 3.4) and the HTTP Artifact
743 binding (see Section 3.6) to transmit request and response messages in a single protocol exchange using
744 two different bindings.

745 3.5.1 Required Information

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

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

748 **Description:** Given below.

749 **Updates:** Effectively replaces the binding aspects of the Browser/POST profile in [SAML 1.1].

750 3.5.2 Overview

751 The HTTP POST binding is intended for cases in which the SAML requester and responder need to
752 communicate using an HTTP user agent (as defined in HTTP 1.1 [RFC2616]) as an intermediary. This
753 may be necessary, for example, if the communicating parties do not share a direct path of communication.
754 It may also be needed if the responder requires an interaction with the user agent in order to fulfill the
755 request, such as when the user agent must authenticate to it.

756 Note that some HTTP user agents may have the capacity to play a more active role in the protocol
757 exchange and may support other bindings that use HTTP, such as the SOAP and Reverse SOAP
758 bindings. This binding assumes nothing apart from the capabilities of a common web browser.

759 3.5.3 RelayState

760 RelayState data MAY be included with a SAML protocol message transmitted with this binding. The value
761 MUST NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the
762 message independent of any other protections that may or may not exist during message transmission.

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

767 If no such value is included with a SAML request message, or if the SAML response message is being

768 generated without a corresponding request, then the SAML responder MAY include RelayStatedata to be
769 interpreted by the recipient based on the use of a profile or prior agreement between the parties.

770 **3.5.4 Message Encoding**

771 Messages are encoded for use with this binding by encoding the XML into an HTML form control and are
772 transmitted using the HTTP POST method. A SAML protocol message is form-encoded by applying the
773 base-64 encoding rules to the XML representation of the message and placing the result in a hidden form
774 control within a form as defined by [HTML401] §17. The HTML document MUST adhere to the XHTML
775 specification, [XHTML] . The base64-encoded value MAY be line-wrapped at a reasonable length in
776 accordance with common practice.

777 If the message is a SAML request, then the form control MUST be named `SAMLRequest`. If the message
778 is a SAML response, then the form control MUST be named `SAMLResponse`. Any additional form controls
779 or presentation MAY be included but MUST NOT be required in order for the recipient to process the
780 message.

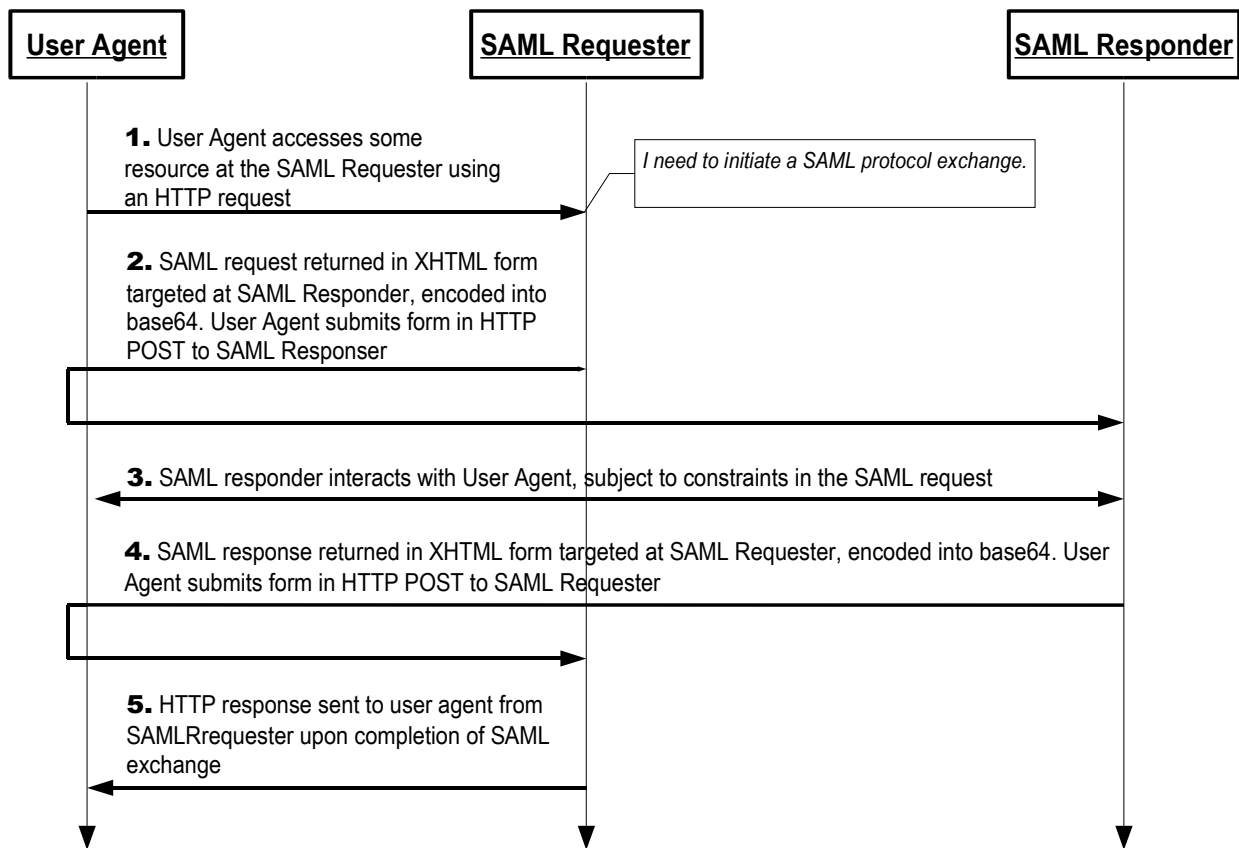
781 If a "RelayState" value is to accompany the SAML protocol message, it MUST be placed in an additional
782 hidden form control named `RelayState` within the same form with the SAML message.

783 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using
784 this binding to which the SAML message is to be delivered. The `method` attribute MUST be "POST".

785 Any technique supported by the user agent MAY be used to cause the submission of the form, and any
786 form content necessary to support this MAY be included, such as submit controls and client-side scripting
787 commands. However, the recipient MUST be able to process the message without regard for the
788 mechanism by which the form submission is initiated.

789 **3.5.5 Message Exchange**

790 The system model used for SAML conversations via this binding is a request-response model, but these
791 messages are sent to the user agent in an HTTP response and delivered to the message recipient in an
792 HTTP request. The HTTP interactions before, between, and after these exchanges take place is
793 unspecified. Both the SAML requester and responder are assumed to be HTTP responders. See the
794 following diagram illustrating the messages exchanged.



- 795 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
796 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 797 2. The system entity acting as a SAML requester responds to an HTTP request from the user agent by
798 returning a SAML request. The request is returned in an [XHTML] document containing the form
799 and content defined in section 3.5.4. The user agent delivers the SAML request by issuing an HTTP
800 POST request to the SAML responder.
- 801 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
802 SAML response or MAY return arbitrary content to facilitate subsequent interaction with the user
803 agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to
804 indicate the requester's level of willingness to permit this kind of interaction (for example, the
805 `IsPassive` attribute in `<samlp:AuthnRequest>`).
- 806 4. Eventually the responder SHOULD return a SAML response to the user agent to be returned to the
807 SAML requester. The SAML response is returned in the same fashion as described for the SAML
808 request in step 2.
- 809 5. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
810 user agent.

811 3.5.5.1 HTTP and Caching Considerations

812 HTTP proxies and the user agent intermediary should not cache SAML protocol messages. To insure this,
813 the following rules SHOULD be followed.

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

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

816 • Include a `Pragma` header field set to "no-cache".

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

818 3.5.5.2 Security Considerations

819 The presence of the user agent intermediary means that the requester and responder cannot rely on the
820 transport layer for end-end authentication, integrity or confidentiality protection. and must authenticate the
821 messages received instead. SAML provides for a signature on protocol messages for authentication and
822 integrity for such cases. Form-encoded messages MAY be signed before the base64 encoding is applied.

823 If the message is signed, the `Destination` XML attribute in the root SAML element of the protocol
824 message MUST contain the URL to which the sender has instructed the user agent to deliver the
825 message. The recipient MUST then verify that the value matches the location at which the message has
826 been received.

827 This binding SHOULD NOT be used if the content of the request or response should not be exposed to
828 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is
829 OPTIONAL and depends on the environment of use. If confidentiality is necessary, SSL 3.0 or TLS 1.0
830 SHOULD be used to protect the message in transit between the user agent and the SAML requester and
831 responder.

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

834 Note also that there is no mechanism defined to protect the integrity of the relationship between the SAML
835 protocol message and the "RelayState" value, if any. That is, an attacker can potentially recombine a pair
836 of valid HTTP responses by switching the "RelayState" values associated with each SAML protocol
837 message. The individual "RelayState" and SAML message values can be integrity protected, but not the
838 combination. As a result, the producer and consumer of "RelayState" information MUST take care not to
839 associate sensitive state information with the "RelayState" value without taking additional precautions
840 (such as based on the information in the SAML message).

841 3.5.6 Error Reporting

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

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

847 For more information about SAML status codes, see the SAML assertions and protocols specification
848 [SAMLCore].

849 3.5.7 Metadata Considerations

850 Support for the HTTP POST binding SHOULD be reflected by indicating URL endpoints at which requests
851 and responses for a particular protocol or profile should be sent. Either a single endpoint or distinct
852 request and response endpoints MAY be supplied.

853 3.5.8 Example SAML Message Exchange Using HTTP POST

854 In this example, a `<LogoutRequest>` and `<LogoutResponse>` message pair are exchanged using the
855 HTTP POST binding.

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

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

973 **Description:** Given below.

974 **Updates:** Effectively replaces the binding aspects of the Browser/Artifact profile in [SAML 1.1].

975 **3.6.2 Overview**

976 The HTTP Artifact binding is intended for cases in which the SAML requester and responder need to
977 communicate using an HTTP user agent as an intermediary, but the intermediary's limitations preclude or
978 discourage the transmission of an entire message (or message exchange) through it. This may be for
979 technical reasons or because of a reluctance to expose the message content to the intermediary (and if
980 the use of encryption is not practical).

981 Note that because of the need to subsequently resolve the artifact using another synchronous binding,
982 such as SOAP, a direct communication path must exist between the SAML message sender and recipient
983 in the reverse direction of the artifact's transmission (the receiver of the message and artifact must be
984 able to send a `<samlp:ArtifactResolve>` request back to the artifact issuer). The artifact issuer must
985 also maintain state while the artifact is pending, which has implications for load-balanced environments.

986 **3.6.3 Message Encoding**

987 There are two methods of encoding an artifact for use with this binding. One is to encode the artifact into a
988 URL parameter and the other is to place the artifact in an HTML form control. When URL encoding is
989 used, the HTTP GET method is used to deliver the message, while POST is used with form encoding. All
990 endpoints that support this binding MUST support both techniques.

991 **3.6.3.1 RelayState**

992 RelayState data MAY be included with a SAML artifact transmitted with this binding. The value MUST
993 NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the message
994 independent of any other protections that may or may not exist during message transmission.

995 If an artifact that represents a SAML request is accompanied by RelayState data, then the SAML
996 responder MUST return its SAML protocol response using a binding that also supports a RelayState
997 mechanism, and it MUST place the exact data it received with the artifact into the corresponding
998 RelayState parameter in the response.

999 If no such value is included with an artifact representing a SAML request, or if the SAML response
1000 message is being generated without a corresponding request, then the SAML responder MAY include
1001 RelayState data to be interpreted by the recipient based on the use of a profile or prior agreement
1002 between the parties.

1003 **3.6.3.2 URL Encoding**

1004 To encode an artifact into a URL, the artifact value is URL-encoded and placed in a query string
1005 parameter named `SAMLart`.

1006 If a "RelayState" value is to accompany the SAML artifact, it MUST be URL-encoded and placed in an
1007 additional query string parameter named `RelayState`.

1008 **3.6.3.3 Form Encoding**

1009 A SAML artifact is form-encoded by placing it in a hidden form control within a form as defined by
1010 [HTML401], chapter 17. The HTML document MUST adhere to the XHTML specification, [XHTML] . The
1011 form control MUST be named `SAMLart`. Any additional form controls or presentation MAY be included but
1012 MUST NOT be required in order for the recipient to process the artifact.

1013 If a "RelayState" value is to accompany the SAML artifact, it MUST be placed in an additional hidden form
1014 control named `RelayState`, within the same form with the SAML message.

1015 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using
1016 this binding to which the artifact is to be delivered. The `method` attribute MUST be set to "POST".

1017 Any technique supported by the user agent MAY be used to cause the submission of the form, and any
1018 form content necessary to support this MAY be included, such as submit controls and client-side scripting
1019 commands. However, the recipient MUST be able to process the artifact without regard for the
1020 mechanism by which the form submission is initiated.

1021 3.6.4 Artifact Format

1022 With respect to this binding, an artifact is a short, opaque string. Different types can be defined and used
1023 without affecting the binding. The important characteristics are the ability of an artifact receiver to identify
1024 the issuer of the artifact, resistance to tampering and forgery, uniqueness, and compactness.

1025 The general format of any artifact includes a mandatory two-byte artifact type code and a two-byte index
1026 value identifying a specific endpoint of the artifact resolution service of the issuer, as follows:

```
1027 SAML_artifact      := B64( TypeCode EndpointIndex RemainingArtifact )
1028 TypeCode           := Byte1Byte2
1029 EndpointIndex      := Byte1Byte2
```

1030 The notation `B64(TypeCode EndpointIndex RemainingArtifact)` stands for the application of
1031 the base64 [RFC2045] transformation to the catenation of the `TypeCode`, `EndpointIndex`, and
1032 `RemainingArtifact`.

1033 The following practices are RECOMMENDED for the creation of SAML artifacts:

- 1034 • Each issuer is assigned an identifying URI, also known as the issuer's entity (or provider) ID. See
1035 section 8.3.6 of [SAMLCore] for a discussion of this kind of identifier.
- 1036 • The issuer constructs the `SourceID` component of the artifact by taking the SHA-1 hash of the
1037 identification URL. The hash value is NOT encoded into hexadecimal.
- 1038 • The `MessageHandle` value is constructed from a cryptographically strong random or
1039 pseudorandom number sequence [RFC1750] generated by the issuer. The sequence consists of
1040 values of at least 16 bytes in size. These values should be padded as needed to a total length of 20
1041 bytes.

1042 The following describes the single artifact type defined by SAML 2.0.

1043 3.6.4.1 Required Information

1044 **Identification:** urn:oasis:names:tc:SAML:2.0:artifact-04

1045 **Contact information:** security-services-comment@lists.oasis-open.org

1046 **Description:** Given below.

1047 **Updates:** None.

1048 3.6.4.2 Format Details

1049 SAML 2.0 defines an artifact type of type code 0x0004. This artifact type is defined as follows:

```
1050 TypeCode           := 0x0004
1051 RemainingArtifact := SourceID MessageHandle
1052 SourceID           := 20-byte_sequence
```

1053 `MessageHandle` := 20-byte_sequence

1054 `SourceID` is a 20-byte sequence used by the artifact receiver to determine artifact issuer identity and the
1055 set of possible resolution endpoints.

1056 It is assumed that the destination site will maintain a table of `SourceID` values as well as one or more
1057 indexed URL endpoints (or addresses) for the corresponding SAML responder. The SAML metadata
1058 specification [SAMLMeta] MAY be used for this purpose. On receiving the SAML artifact, the receiver
1059 determines if the `SourceID` belongs to a known artifact issuer and obtains the location of the SAML
1060 responder using the `EndpointIndex` before sending a SAML `<samlp:ArtifactResolve>` message
1061 to it.

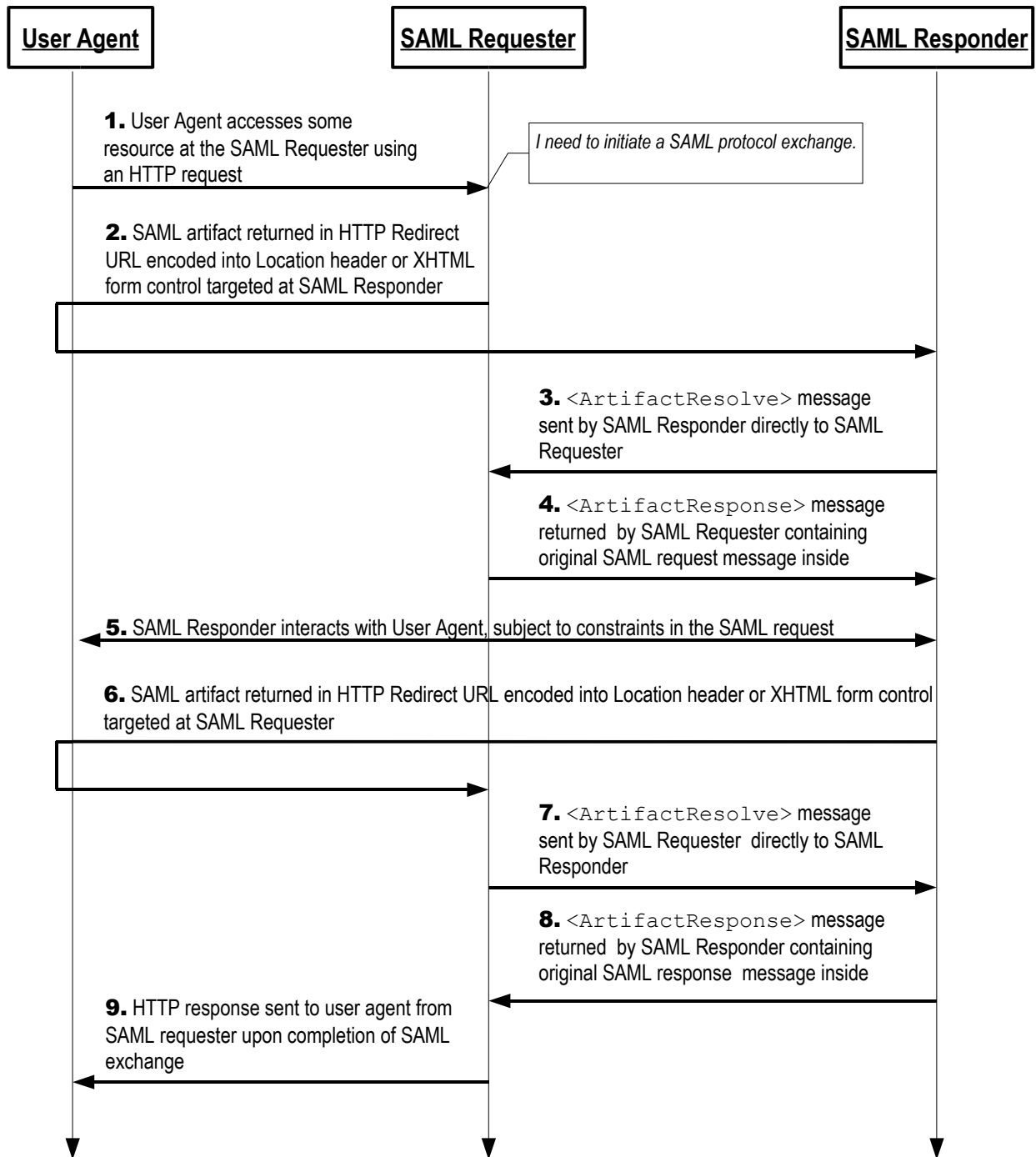
1062 Any two artifact issuers with a common receiver MUST use distinct `SourceID` values. Construction of
1063 `MessageHandle` values is governed by the principle that they SHOULD have no predictable relationship
1064 to the contents of the referenced message at the issuing site and it MUST be infeasible to construct or
1065 guess the value of a valid, outstanding message handle.

1066 **3.6.5 Message Exchange**

1067 The system model used for SAML conversations by means of this binding is a request-response model in
1068 which an artifact reference takes the place of the actual message content, and the artifact reference is
1069 sent to the user agent in an HTTP response and delivered to the message recipient in an HTTP request.
1070 The HTTP interactions before, between, and after these exchanges take place is unspecified. Both the
1071 SAML requester and responder are assumed to be HTTP responders.

1072 Additionally, it is assumed that on receipt of an artifact by way of the user agent, the recipient invokes a
1073 separate, direct exchange with the artifact issuer using the Artifact Resolution Protocol defined in
1074 [SAMLCore]. This exchange MUST use a binding that does not use the HTTP user agent as an
1075 intermediary, such as the SOAP binding. On the successful acquisition of a SAML protocol message, the
1076 artifact is discarded and the processing of the primary SAML protocol exchange resumes (or ends, if the
1077 message is a response).

1078 Issuing and delivering an artifact, along with the subsequent resolution step, constitutes half of the overall
1079 SAML protocol exchange. This binding can be used to deliver either or both halves of a SAML protocol
1080 exchange. A binding composable with it, such as the HTTP Redirect (see Section 3.4) or POST (see
1081 Section 3.5) binding, MAY be used to carry the other half of the exchange. The following sequence
1082 assumes that the artifact binding is used for both halves. See the diagram below illustrating the messages
1083 exchanged.



- 1084 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
1085 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 1086 2. The system entity acting as a SAML requester responds to an HTTP request from the user agent by
1087 returning an artifact representing a SAML request.
- 1088 • If URL-encoded, the artifact is returned encoded into the HTTP response's Location
1089 header, and the HTTP status MUST be either 303 or 302. The SAML requester MAY
1090 include additional presentation and content in the HTTP response to facilitate the user
1091 agent's transmission of the message, as defined in HTTP 1.1 [RFC2616]. The user

- 1092 agent delivers the artifact by issuing an HTTP GET request to the SAML responder.
- 1093 • If form-encoded, then the artifact is returned in an XHTML document containing the
1094 form and content defined in Section 3.6.3.3. The user agent delivers the artifact by
1095 issuing an HTTP POST request to the SAML responder.
- 1096 3. The SAML responder determines the SAML requester by examining the artifact (the exact process
1097 depends on the type of artifact), and issues a `<samlp:ArtifactResolve>` request containing
1098 the artifact to the SAML requester using a direct SAML binding, temporarily reversing roles.
- 1099 4. Assuming the necessary conditions are met, the SAML requester returns a
1100 `<samlp:ArtifactResponse>` containing the original SAML request message it wishes the
1101 SAML responder to process.
- 1102 5. In general, the SAML responder MAY respond to the SAML request by immediately returning a
1103 SAML artifact or MAY return arbitrary content to facilitate subsequent interaction with the user agent
1104 necessary to fulfill the request. Specific protocols and profiles may include mechanisms to indicate
1105 the requester's level of willingness to permit this kind of interaction (for example, the `IsPassive`
1106 attribute in `<samlp:AuthnRequest>`).
- 1107 6. Eventually the responder SHOULD return a SAML artifact to the user agent to be returned to the
1108 SAML requester. The SAML response artifact is returned in the same fashion as described for the
1109 SAML request artifact in step 2. The SAML requester determines the SAML responder by examining
1110 the artifact, and issues a `<samlp:ArtifactResolve>` request containing the artifact to the SAML
1111 responder using a direct SAML binding, as in step 3.
- 1112 7. Assuming the necessary conditions are met, the SAML responder returns a
1113 `<samlp:ArtifactResponse>` containing the SAML response message it wishes the requester to
1114 process, as in step 4.
- 1115 8. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
1116 user agent.

1117 **3.6.5.1 HTTP and Caching Considerations**

1118 HTTP proxies and the user agent intermediary should not cache SAML artifacts. To insure this, the
1119 following rules SHOULD be followed.

1120 When returning SAML artifacts using HTTP 1.1, HTTP responders SHOULD:

- 1121 • Include a `Cache-Control` header field set to "no-cache, no-store".
1122 • Include a `Pragma` header field set to "no-cache".

1123 There are no other restrictions on the use of HTTP headers.

1124 **3.6.5.2 Security Considerations**

1125 This binding uses a combination of indirect transmission of a message reference followed by a direct
1126 exchange to return the actual message. As a result, the message reference (artifact) need not itself be
1127 authenticated or integrity protected, but the callback request/response exchange that returns the actual
1128 message MAY be mutually authenticated and integrity protected, depending on the environment of use.

1129 If the actual SAML protocol message is intended for a specific recipient, then the artifact's issuer MUST
1130 authenticate the sender of the subsequent `<samlp:ArtifactResolve>` message before returning the
1131 actual message.

1132 The transmission of an artifact to and from the user agent SHOULD be protected with confidentiality; SSL
1133 3.0 or TLS 1.0 SHOULD be used. The callback request/response exchange that returns the actual
1134 message MAY be protected, depending on the environment of use.

1135 In general, this binding relies on the artifact as a hard-to-forge short-term reference and applies other
1136 security measures to the callback request/response that returns the actual message. All artifacts MUST
1137 have a single-use semantic enforced by the artifact issuer.

1138 Furthermore, it is RECOMMENDED that artifact receivers also enforce a single-use semantic on the
1139 artifact values they receive, to prevent an attacker from interfering with the resolution of an artifact by a
1140 user agent and then resubmitting it to the artifact receiver. If an attempt to resolve an artifact does not
1141 complete successfully, the artifact SHOULD be placed into a blocked artifact list for a period of time that
1142 exceeds a reasonable acceptance period during which the artifact issuer would resolve the artifact.

1143 Note also that there is no mechanism defined to protect the integrity of the relationship between the
1144 artifact and the "RelayState" value, if any. That is, an attacker can potentially recombine a pair of valid
1145 HTTP responses by switching the "RelayState" values associated with each artifact. As a result, the
1146 producer/consumer of "RelayState" information MUST take care not to associate sensitive state
1147 information with the "RelayState" value without taking additional precautions (such as based on the
1148 information in the SAML protocol message retrieved via artifact).

1149 **3.6.6 Error Reporting**

1150 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD
1151 return a response message with a second-level `<samlp:StatusCode>` value of
1152 `urn:oasis:names:tc:SAML:2.0:status:RequestDenied`.

1153 HTTP interactions during the message exchange MUST NOT use HTTP error status codes to indicate
1154 failures in SAML processing, since the user agent is not a full party to the SAML protocol exchange.

1155 If the issuer of an artifact receives a `<samlp:ArtifactResolve>` message that it can understand, it
1156 MUST return a `<samlp:ArtifactResponse>` with a `<samlp:StatusCode>` value of
1157 `urn:oasis:names:tc:SAML:2.0:status:Success`, even if it does not return the corresponding
1158 message (for example because the artifact requester is not authorized to receive the message or the
1159 artifact is no longer valid).

1160 For more information about SAML status codes, see the SAML assertions and protocols specification
1161 [SAMLCore].

1162 **3.6.7 Metadata Considerations**

1163 Support for the HTTP Artifact binding SHOULD be reflected by indicating URL endpoints at which
1164 requests and responses for a particular protocol or profile should be sent. Either a single endpoint or
1165 distinct request and response endpoints MAY be supplied. One or more indexed endpoints for processing
1166 `<samlp:ArtifactResolve>` messages SHOULD also be described.

1167 **3.6.8 Example SAML Message Exchange Using HTTP Artifact**

1168 In this example, a `<LogoutRequest>` and `<LogoutResponse>` message pair are exchanged using the
1169 HTTP Artifact binding, with the artifact resolution taking place using the SOAP binding bound to HTTP.

1170 First, here are the actual SAML protocol messages being exchanged:

```
1171 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
1172 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
1173 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
1174 21T19:00:49Z" Version="2.0">  
1175 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
1176 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
1177 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
1178 <samlp:SessionIndex>1</samlp:SessionIndex>  
1179 </samlp:LogoutRequest>
```

```

1180 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1181 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1182 ID="b0730d21b628110d8b7e004005b13a2b"
1183 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"
1184 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">
1185 <Issuer>https://ServiceProvider.com/SAML</Issuer>
1186 <samlp:Status>
1187 <samlp:StatusCode
1188 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1189 </samlp:Status>
1190 </samlp:LogoutResponse>

```

1191 The initial HTTP request from the user agent in step 1 is not defined by this binding. To initiate the logout
 1192 protocol exchange, the SAML requester returns the following HTTP response, containing a SAML artifact.
 1193 Note that the line feeds in the HTTP Location header below are a result of document formatting, and
 1194 there are no line feeds in the actual header value.

```

1195 HTTP/1.1 302 Object Moved
1196 Date: 21 Jan 2004 07:00:49 GMT
1197 Location:
1198 https://ServiceProvider.com/SAML/SLO/Browser?SAMLart=AAQAADWNEw5VT47wcO4z
1199 X%2FiEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU%
1200 3D&RelayState=0043bfclbc45110dae17004005b13a2b
1201 Content-Type: text/html; charset=iso-8859-1

```

1202 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact
 1203 Resolution protocol and the SOAP binding in steps 3 and 4, as follows:

1204 Step 3:

```

1205 POST /SAML/Artifact/Resolve HTTP/1.1
1206 Host: IdentityProvider.com
1207 Content-Type: text/xml
1208 Content-Length: nnn
1209 SOAPAction: http://www.oasis-open.org/committees/security
1210 <SOAP-ENV:Envelope
1211 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1212 <SOAP-ENV:Body>
1213 <samlp:ArtifactResolve
1214 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1215 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1216 ID="_6c3a4f8b9c2d" Version="2.0"
1217 IssueInstant="2004-01-21T19:00:49Z">
1218 <Issuer>https://ServiceProvider.com/SAML</Issuer>
1219 <Artifact>
1220 AAQAADWNEw5VT47wcO4zX/iEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU=
1221 </Artifact>
1222 </samlp:ArtifactResolve>
1223 </SOAP-ENV:Body>
1224 </SOAP-ENV:Envelope>

```

1225 Step 4:

```

1226 HTTP/1.1 200 OK
1227 Date: 21 Jan 2004 07:00:49 GMT
1228 Content-Type: text/xml
1229 Content-Length: nnnn
1230 <SOAP-ENV:Envelope
1231 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1232 <SOAP-ENV:Body>
1233 <samlp:ArtifactResponse
1234 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1235 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1236 ID="FQvGknDfws2Z" Version="2.0"
1237 InResponseTo="_6c3a4f8b9c2d"
1238 IssueInstant="2004-01-21T19:00:49Z">
1239 <Issuer>https://IdentityProvider.com/SAML</Issuer>
1240 <samlp:Status>

```

```

1241         <samlp:StatusCode
1242         Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1243     </samlp:Status>
1244     <samlp:LogoutRequest ID="d2b7c388cec36fa7c39c28fd298644a8"
1245     IssueInstant="2004-01-21T19:00:49Z"
1246     Version="2.0">
1247         <Issuer>https://IdentityProvider.com/SAML</Issuer>
1248         <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
1249 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>
1250         <samlp:SessionIndex>1</samlp:SessionIndex>
1251     </samlp:LogoutRequest>
1252 </samlp:ArtifactResponse>
1253 </SOAP-ENV:Body>
1254 </SOAP-ENV:Envelope>

```

1255 After any unspecified interactions may have taken place, the SAML responder returns a second SAML
1256 artifact in its HTTP response in step 6:

```

1257 HTTP/1.1 302 Object Moved
1258 Date: 21 Jan 2004 07:05:49 GMT
1259 Location:
1260 https://IdentityProvider.com/SAML/SLO/Response?SAMLart=AAQAAFQIZXv5%
1261 2BQaBaE5qYurHWJO1nAgLASqfnyidHIggbFU0mlSGFTyQiPc%
1262 3D&RelayState=0043bfc1bc45110dael7004005b13a2b
1263 Content-Type: text/html; charset=iso-8859-1

```

1264 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact
1265 Resolution protocol and the SOAP binding in steps 7 and 8, as follows:

1266 Step 7:

```

1267 POST /SAML/Artifact/Resolve HTTP/1.1
1268 Host: ServiceProvider.com
1269 Content-Type: text/xml
1270 Content-Length: nnn
1271 SOAPAction: http://www.oasis-open.org/committees/security
1272 <SOAP-ENV:Envelope
1273     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1274     <SOAP-ENV:Body>
1275         <samlp:ArtifactResolve
1276             xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1277             xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1278             ID="_ec36fa7c39" Version="2.0"
1279             IssueInstant="2004-01-21T19:05:49Z">
1280             <Issuer>https://IdentityProvider.com/SAML</Issuer>
1281             <Artifact>
1282             AAQAAFQIZXv5+QaBaE5qYurHWJO1nAgLASqfnyidHIggbFU0mlSGFTyQiPc=
1283             </Artifact>
1284         </samlp:ArtifactResolve>
1285     </SOAP-ENV:Body>
1286 </SOAP-ENV:Envelope>

```

1287 Step 8:

```

1288 HTTP/1.1 200 OK
1289 Date: 21 Jan 2004 07:05:49 GMT
1290 Content-Type: text/xml
1291 Content-Length: nnnn
1292 <SOAP-ENV:Envelope
1293     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1294     <SOAP-ENV:Body>
1295         <samlp:ArtifactResponse
1296             xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1297             xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1298             ID="_FQvGknDfws2Z" Version="2.0"
1299             InResponseTo="_ec36fa7c39"
1300             IssueInstant="2004-01-21T19:05:49Z">

```

```

1301     <Issuer>https://ServiceProvider.com/SAML</Issuer>
1302     <samlp:Status>
1303       <samlp:StatusCode
1304         Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1305     </samlp:Status>
1306     <samlp:LogoutResponse ID="_b0730d21b628110d8b7e004005b13a2b"
1307       InResponseTo="_d2b7c388cec36fa7c39c28fd298644a8"
1308       IssueInstant="2004-01-21T19:05:49Z"
1309       Version="2.0">
1310       <Issuer>https://ServiceProvider.com/SAML</Issuer>
1311       <samlp:Status>
1312         <samlp:StatusCode
1313           Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1314         </samlp:Status>
1315       </samlp:LogoutResponse>
1316     </samlp:ArtifactResponse>
1317   </SOAP-ENV:Body>
1318 </SOAP-ENV:Envelope>

```

1319 3.7 SAML URI Binding

1320 URIs are a protocol-independent means of referring to a resource. This binding is not a general SAML
 1321 request/response binding, but rather supports the encapsulation of a <samlp:AssertionIDRequest>
 1322 message with a single <saml:AssertionIDRef> into the resolution of a URI. The result of a successful
 1323 request is a SAML <saml:Assertion> element (but not a complete SAML response).

1324 Like SOAP, URI resolution can occur over multiple underlying transports. This binding has transport-
 1325 independent aspects, but also calls out the use of HTTP with SSL 3.0 or TLS 1.0 as REQUIRED
 1326 (mandatory to implement).

1327 3.7.1 Required Information

1328 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:URI

1329 **Contact information:** security-services-comment@lists.oasis-open.org

1330 **Description:** Given below.

1331 **Updates:** None

1332 3.7.2 Protocol-Independent Aspects of the SAML URI Binding

1333 The following sections define aspects of the SAML URI binding that are independent of the underlying
 1334 transport protocol of the URI resolution process.

1335 3.7.2.1 Basic Operation

1336 A SAML URI reference identifies a specific SAML assertion. The result of resolving the URI MUST be a
 1337 message containing the assertion, or a transport-specific error. The specific format of the message
 1338 depends on the underlying transport protocol. If the transport protocol permits the returned content to be
 1339 described, such as HTTP 1.1 [RFC2616], then the assertion MAY be encoded in whatever format is
 1340 permitted. If not, the assertion MUST be returned in a form which can be unambiguously interpreted as or
 1341 transformed into an XML serialization of the assertion.

1342 It MUST be the case that if the same URI reference is resolved in the future, then either the same SAML
 1343 assertion, or an error, is returned. That is, the reference MAY be persistent but MUST consistently
 1344 reference the same assertion, if any.

1345 **3.7.3 Security Considerations**

1346 Indirect use of a SAML assertion presents dangers if the binding of the reference to the result is not
1347 secure. The particular threats and their severity depend on the use to which the assertion is being put. In
1348 general, the result of resolving a URI reference to a SAML assertion SHOULD only be trusted if the
1349 requester can be certain of the identity of the responder and that the contents have not been modified in
1350 transit.

1351 It is often not sufficient that the assertion itself be signed, because URI references are by their nature
1352 somewhat opaque to the requester. The requester SHOULD have independent means to insure that the
1353 assertion returned is actually the one that is represented by the URI; this is accomplished by both
1354 authenticating the responder and relying on the integrity of the response.

1355 **3.7.4 MIME Encapsulation**

1356 For resolution protocols that support MIME as a content description and packaging mechanism, the
1357 resulting assertion SHOULD be returned as a MIME entity of type `application/samlassertion+xml`,
1358 as defined by [\[SAMLmime\]](#).

1359 **3.7.5 Use of HTTP URIs**

1360 A SAML authority that claims conformance to the SAML URI binding MUST implement support for HTTP.
1361 This section describes certain specifics of using HTTP URIs, including URI syntax, HTTP headers, and
1362 error reporting.

1363 **3.7.5.1 URI Syntax**

1364 In general, there are no restrictions on the permissible syntax of a SAML URI reference as long as the
1365 SAML authority responsible for the reference creates the message containing it. However, authorities
1366 MUST support a URL endpoint at which an HTTP request can be sent with a single query string
1367 parameter named `ID`. There MUST be no query string in the endpoint URL itself independent of this
1368 parameter.

1369 For example, if the documented endpoint at an authority is "<https://saml.example.edu/assertions>", a
1370 request for an assertion with an `ID` of `abcde` can be sent to:

1371 <https://saml.example.edu/assertions?ID=abcde>

1372 Note that the use of wildcards is not allowed for such ID queries.

1373 **3.7.5.2 HTTP and Caching Considerations**

1374 HTTP proxies MUST NOT cache SAML assertions. To insure this, the following rules SHOULD be
1375 followed.

1376 When returning SAML assertions using HTTP 1.1, HTTP responders SHOULD:

- 1377 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 1378 • Include a `Pragma` header field set to "no-cache".

1379 **3.7.5.3 Security Considerations**

1380 [RFC2617] describes possible attacks in the HTTP environment when basic or message-digest
1381 authentication schemes are used.

1382 Use of SSL 3.0 or TLS 1.0 is STRONGLY RECOMMENDED as a means of authentication, integrity
1383 protection, and confidentiality.

1384 **3.7.5.4 Error Reporting**

1385 As an HTTP protocol exchange, the appropriate HTTP status code SHOULD be used to indicate the result
1386 of a request. For example, a SAML responder that refuses to perform a message exchange with the
1387 SAML requester SHOULD return a "403 Forbidden" response. If the assertion specified is unknown to
1388 the responder, then a "404 Not Found" response SHOULD be returned. In these cases, the content of
1389 the HTTP body is not significant.

1390 **3.7.5.5 Metadata Considerations**

1391 Support for the URI binding over HTTP SHOULD be reflected by indicating a URL endpoint at which
1392 requests for arbitrary assertions are to be sent.

1393 **3.7.5.6 Example SAML Message Exchange Using an HTTP URI**

1394 Following is an example of a request for an assertion.

```
1395 GET /SamlService?ID=abcde HTTP/1.1  
1396 Host: www.example.com
```

1397 Following is an example of the corresponding response, which supplies the requested assertion.

```
1398 HTTP/1.1 200 OK  
1399 Content-Type: application/samlassertion+xml  
1400 Cache-Control: no-cache, no-store  
1401 Pragma: no-cache  
1402 Content-Length: nnnn  
  
1403 <saml:Assertion ID="abcde" ...>  
1404 ...  
1405 </saml:Assertion>
```

4 References

1406

- 1407 **[AES]** FIPS-197, Advanced Encryption Standard (AES), available from
1408 <http://www.nist.gov/>.
- 1409 **[Anders]** A suggestion on how to implement SAML browser bindings without using
1410 “Artifacts”, <http://www.x-obi.com/OBI400/andersr-browser-artifact.ppt>.
- 1411 **[CoreAssnEx]** Core Assertions Architecture, Examples and Explanations, [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-phill-07.pdf)
1412 [open.org/committees/security/docs/draft-sstc-core-phill-07.pdf](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-phill-07.pdf).
- 1413 **[HTML401]** HTML 4.01 Specification, W3C Recommendation 24 December 1999,
1414 <http://www.w3.org/TR/html4>.
- 1415 **[XHTML]** XHTML 1.0 The Extensible HyperText Markup Language (Second Edition),
1416 <http://www.w3.org/TR/xhtml1/>.
- 1417 **[Liberty]** The Liberty Alliance Project, <http://www.projectliberty.org>.
- 1418 **[MSURL]** Microsoft technical support article,
1419 <http://support.microsoft.com/support/kb/articles/Q208/4/27.ASP>.
- 1420 **[PAOS]** Aarts, R., “Liberty Reverse HTTP Binding for SOAP Specification”, Version: 1.0,
1421 <https://www.projectliberty.org/specs/liberty-paos-v1.0.pdf>
- 1422 **[Rescorla-Sec]** E. Rescorla et al., *Guidelines for Writing RFC Text on Security Considerations*,
1423 <http://www.ietf.org/internet-drafts/draft-iab-sec-cons-03.txt>.
- 1424 **[RFC1952]** GZIP file format specification version 4.3, <http://www.ietf.org/rfc/rfc1952.txt>
- 1425 **[RFC1738]** Uniform Resource Locators (URL), <http://www.ietf.org/rfc/rfc1738.txt>
- 1426 **[RFC1750]** Randomness Recommendations for Security. <http://www.ietf.org/rfc/rfc1750.txt>
- 1427 **[RFC1945]** Hypertext Transfer Protocol -- HTTP/1.0, <http://www.ietf.org/rfc/rfc1945.txt>.
- 1428 **[RFC2045]** Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet
1429 Message Bodies, <http://www.ietf.org/rfc/rfc2045.txt>
- 1430 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF
1431 RFC 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>.
- 1432 **[RFC2246]** The TLS Protocol Version 1.0, <http://www.ietf.org/rfc/rfc2246.txt>.
- 1433 **[RFC2279]** UTF-8, a transformation format of ISO 10646, <http://www.ietf.org/rfc/rfc2279.txt>.
- 1434 **[RFC2616]** Hypertext Transfer Protocol -- HTTP/1.1, <http://www.ietf.org/rfc/rfc2616.txt>.
- 1435 **[RFC2617]** *HTTP Authentication: Basic and Digest Access Authentication*, IETF RFC 2617,
1436 <http://www.ietf.org/rfc/rfc2617.txt>.
- 1437 **[SAMLCore]** S. Cantor et al., *Assertions and Protocols for the OASIS Security Assertion*
1438 *Markup Language (SAML) V2.0*. OASIS SSTC, September 2004. Document ID
1439 sstc-saml-core-2.0-cd-02. See <http://www.oasis-open.org/committees/security/>.
- 1440 **[SAMLGloss]** J. Hodges et al., *Glossary for the OASIS Security Assertion Markup Language*
1441 *(SAML) V2.0*. OASIS SSTC, September 2004. Document ID sstc-saml-glossary-
1442 2.0-cd-02. See <http://www.oasis-open.org/committees/security/>.
- 1443 **[SAMLProfile]** S. Cantor et al., *Profiles for the OASIS Security Assertion Markup Language*
1444 *(SAML) V2.0*. OASIS SSTC, September 2004. Document ID sstc-saml-profiles-
1445 2.0-cd-02. See <http://www.oasis-open.org/committees/security/>.
- 1446 **[SAMLMeta]** S. Cantor et al., *Metadata for the OASIS Security Assertion Markup Language*
1447 *(SAML) V2.0*. OASIS SSTC, September 2004. Document ID sstc-saml-metadata-
1448 2.0-cd-02. See <http://www.oasis-open.org/committees/security/>.
- 1449 **[SAMLmime]** application/saml+xml Media Type Registration, IETF Internet-Draft,
1450 <http://www.ietf.org/internet-drafts/draft-hodges-saml-mediatype-01.txt>.

1451	[SAMLSecure]	F. Hirsch et al., <i>Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0</i> . OASIS SSTC, September 2004. Document ID sstc-saml-sec-consider-2.0-cd-02. See http://www.oasis-open.org/committees/security/ .
1452		
1453		
1454		
1455	[SAMLReqs]	Darren Platt et al., <i>SAML Requirements and Use Cases</i> , OASIS, April 2002, http://www.oasis-open.org/committees/security/ .
1456		
1457	[SAMLWeb]	OASIS Security Services Technical Committee website, http://www.oasis-open.org/committees/security/ .
1458		
1459	[SESSION]	RL "Bob" Morgan, Support of target web server sessions in Shibboleth, http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-session-00.txt
1460		
1461		
1462	[ShibMarlena]	Marlena Erdos et al., <i>Shibboleth Architecture DRAFT v1.1</i> , http://shibboleth.internet2.edu/draft-internet2-shibboleth-arch-v05.html .
1463		
1464	[SOAP1.1]	D. Box et al., <i>Simple Object Access Protocol (SOAP) 1.1</i> , World Wide Web Consortium Note, May 2000, http://www.w3.org/TR/SOAP .
1465		
1466	[SOAP-PRIMER]	N. Mitra, <i>SOAP Version 1.2 Part 0: Primer</i> , W3C Recommendation 24 June 2003, http://www.w3.org/TR/soap12-part0/
1467		
1468	[SSL3]	A. Frier et al., <i>The SSL 3.0 Protocol</i> , Netscape Communications Corp, November 1996.
1469		
1470	[WEBSSO]	RL "Bob" Morgan, Interactions between Shibboleth and local-site web sign-on services, http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-websso-00.txt
1471		
1472		
1473	[WSS-SAML]	P. Hallam-Baker et al., <i>Web Services Security: SAML Token Profile</i> , OASIS, March 2003, http://www.oasis-open.org/committees/wss .
1474		
1475	[WSS-Sec]	A. Nadalin et al., <i>Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)</i> , OASIS Standard 200401, March 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
1476		
1477		
1478	[XMLSig]	D. Eastlake et al., <i>XML-Signature Syntax and Processing</i> , World Wide Web Consortium, http://www.w3.org/TR/xmlsig-core/ .
1479		

1480 5 Registration of MIME media type 1481 application/samlassertion+xml

1482 To: ietf-types@iana.org

1483 Subject: Registration of MIME media type application/samlassertion+xml

1484 Introduction

1485 This document defines a MIME media type -- application/samlassertion+xml -- for use with the
1486 XML serialization of SAML (Security Assertion Markup Language) assertions.

1487 The SAML specification sets -- [SAMLv1.0], [SAMLv1.1], [SAMLv2.0] -- are work products of the
1488 OASIS Security Services Technical Committee [SSTC]. The SAML specifications define XML-
1489 based constructs with which one may make, and convey, security assertions. Using SAML, one
1490 can assert that an authentication event pertaining to some subject has occurred and convey said
1491 assertion to a relying party, for example.

1492 SAML assertions, which are explicitly versioned, are defined by [SAMLv1Core], [SAMLv11Core],
1493 and [SAMLv2Core].

1494 MIME media type name: application

1495 MIME subtype name: samlassertion+xml

1496 Required parameters: none

1497 Optional parameters: charset

1498 Same as charset parameter of application/xml [RFC3023].

1499 Encoding considerations:

1500 Same as for application/xml [RFC3023].

1501 Security considerations:

1502 Per their specification, samlassertion+xml typed objects do not contain executable content.

1503 However, SAML assertions are XML-based objects [XML]. As such, they have all of the general
1504 security considerations presented in section 10 of [RFC3023], as well as additional ones, since
1505 they are explicit security objects. For example, samlassertion+xml typed objects will often contain
1506 data that may identify or pertain to a natural person, and may be used as a basis for sessions and
1507 access control decisions.

1508 To counter potential issues, samlassertion+xml typed objects contain data that should be signed
1509 appropriately by the sender. Any such signature must be verified by the recipient of the data - both
1510 as a valid signature, and as being the signature of the sender. Issuers of samlassertion+xml
1511 objects containing SAMLv2 assertions may also encrypt all, or portions of, the assertions
1512 [SAMLv2Core].

1513 In addition, SAML profiles and protocol bindings specify use of secure channels as appropriate.

1514 [SAMLv2.0] incorporates various privacy-protection techniques in its design. For example: opaque
1515 handles, specific to interactions between specific system entities, may be assigned to subjects.
1516 The handles are mappable to wider-context identifiers (e.g. email addresses, account identifiers,
1517 etc) by only the specific parties.

1518 For a more detailed discussion of SAML security considerations and specific security-related
1519 design techniques, please refer to the SAML specifications listed in the below bibliography. The
1520 specifications containing security-specific information have been explicitly listed for each version

1521 of SAML.

1522 Interoperability considerations:

1523 SAML assertions are explicitly versioned. Relying parties should ensure that they observe
1524 assertion version information and behave accordingly. See "Chapter 4 SAML Versioning" in
1525 [SAMLv1Core], [SAMLv11Core], or [SAMLv2Core], as appropriate.

1526 Published specification:

1527 [SAMLv2Bind] explicitly specifies use of the application/samlassertion+xml MIME media type.
1528 However, it is conceivable that non-SAMLv2 assertions (i.e. SAMLv1 and/or SAMLv1.1) might in
1529 practice be conveyed using SAMLv2 bindings.

1530 Applications which use this media type:

1531 Potentially any application implementing SAML, as well as those applications implementing
1532 specifications based on SAML, e.g. those available from the Liberty Alliance [LAP].

1533 Additional information:

1534 Magic number(s):

1535 In general, the same as for application/xml [RFC3023]. In particular, the XML root
1536 element of the returned object will have a namespace-qualified name with:

1537 – a local name of: Assertion

1538 – a namespace URI of: one of the version-specific SAML assertion XML
1539 namespace URIs, as defined by the appropriate version-specific SAML "core"
1540 specification (see bibliography).

1541 With SAMLv2.0 specifically, the root element of the returned object may be either
1542 <saml:Assertion> or <saml:EncryptedAssertion>, where "saml" represents any XML
1543 namespace prefix that maps to the SAMLv2.0 assertion namespace URI:

1544 urn:oasis:names:tc:SAML:2.0:assertion

1545 File extension(s): none

1546 Macintosh File Type Code(s): none

1547 Person & email address to contact for further information:

1548 This registration is made on behalf of the OASIS Security Services Technical Committee (SSTC)
1549 Please refer to the SSTC website for current information on committee chairperson(s) and their
1550 contact addresses: <http://www.oasis-open.org/committees/security/>. Committee members should
1551 submit comments and potential errata to the securityservices@lists.oasis-open.org list. Others
1552 should submit them by filling out the web form located at [http://www.oasis-
1553 open.org/committees/comments/form.php?wg_abbrev=security](http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=security).

1554 Additionally, the SAML developer community email distribution list, [saml-dev@lists.oasis-
1555 open.org](mailto:saml-dev@lists.oasis-open.org), may be employed to discuss usage of the application/samlassertion+xml MIME media
1556 type. The "saml-dev" mailing list is publicly archived here: [http://lists.oasis-
1557 open.org/archives/saml-dev/](http://lists.oasis-open.org/archives/saml-dev/). To post to the "saml-dev" mailing list, one must subscribe to it. To
1558 subscribe, send a message with the single word "subscribe" in the message body, to: [saml-dev-
1559 request@lists.oasis-open.org](mailto:saml-dev-request@lists.oasis-open.org).

1560 Intended usage: COMMON

1561 Author/Change controller:

1562 The SAML specification sets are a work product of the OASIS Security Services Technical
1563 Committee (SSTC). OASIS and the SSTC have change control over the SAML specification sets.

1564 Bibliography

- 1565 [LAP] “*Liberty Alliance Project*”. See <http://www.projectliberty.org/>
- 1566 [OASIS] “*Organization for the Advancement of Structured Information Systems*”.
1567 See <http://www.oasis-open.org/>
- 1568 [RFC3023] M. Murata, S. St.Laurent, D. Kohn, “*XML Media Types*”, IETF Request for
1569 Comments 3023, January 2001. Available as [http://www.rfc-
1570 editor.org/rfc/rfc3023.txt](http://www.rfc-editor.org/rfc/rfc3023.txt)
- 1571 [SAMLv1.0] OASIS Security Services Technical Committee, “*Security Assertion
1572 Markup Language (SAML) Version 1.0 Specification Set*”. OASIS
1573 Standard 200205, November 2002. Available as [http://www.oasis-
1574 open.org/committees/download.php/2290/oasis-sstc-saml-1.0.zip](http://www.oasis-open.org/committees/download.php/2290/oasis-sstc-saml-1.0.zip)
- 1575 [SAMLv1Bind] Prateek Mishra et al., “*Bindings and Profiles for the OASIS Security
1576 Assertion Markup Language (SAML)*”, OASIS, November 2002.
1577 Document ID oasis-sstc-saml-bindings-1.0. See [http://www.oasis-
1578 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1579 [SAMLv1Core] Phillip Hallam-Baker et al., “*Assertions and Protocol for the OASIS
1580 Security Assertion Markup Language (SAML)*”, OASIS, November 2002.
1581 Document ID oasis-sstc-saml-core-1.0. See [http://www.oasis-
1582 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1583 [SAMLv1Sec] Chris McLaren et al., “*Security Considerations for the OASIS Security
1584 Assertion Markup Language (SAML)*”, OASIS, November 2002.
1585 Document ID oasis-sstc-saml-sec-consider-1.0. See [http://www.oasis-
1586 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1587 [SAMLv1.1] OASIS Security Services Technical Committee, “*Security Assertion
1588 Markup Language (SAML) Version 1.1 Specification Set*”. OASIS
1589 Standard 200308, August 2003. Available as [http://www.oasis-
1590 open.org/committees/download.php/3400/oasis-sstc-saml-1.1-pdf-xsd.zip](http://www.oasis-open.org/committees/download.php/3400/oasis-sstc-saml-1.1-pdf-xsd.zip)
- 1591 [SAMLv11Bind] E. Maler et al. “*Bindings and Profiles for the OASIS Security Assertion
1592 Markup Language (SAML)*”. OASIS, September 2003. Document ID
1593 oasis-sstc-saml-bindings-1.1. [http://www.oasis-
1594 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1595 [SAMLv11Core] E. Maler et al. “*Assertions and Protocol for the OASIS Security Assertion
1596 Markup Language (SAML)*”. OASIS, September 2003. Document ID
1597 oasis-sstc-saml-core-1.1. <http://www.oasis-open.org/committees/security/>
- 1598 [SAMLv11Sec] E. Maler et al. “*Security Considerations for the OASIS Security Assertion
1599 Markup Language (SAML)*”. OASIS, September 2003. Document ID
1600 oasis-sstc-saml-sec-consider-1.1. [http://www.oasis-
1601 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1602 [SAMLv2.0] OASIS Security Services Technical Committee, “*Security Assertion
1603 Markup Language (SAML) Version 2.0 Specification Set*”. WORK IN
1604 PROGRESS. Available at [http://www.oasis-
1605 open.org/committees/security/](http://www.oasis-open.org/committees/security/)

1606	[SAMLv2Bind]	S. Cantor et al., " <i>Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0</i> ". OASIS SSTC, August 2004. Document ID sstc-saml-bindings-2.0-cd-01, WORK IN PROGRESS. See http://www.oasis-open.org/committees/security/
1607		
1608		
1609		
1610	[SAMLv2Core]	S. Cantor et al., " <i>Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0</i> ". OASIS SSTC, August 2004. Document ID sstc-saml-core-2.0-cd-01, WORK IN PROGRESS. See http://www.oasis-open.org/committees/security/
1611		
1612		
1613		
1614	[SAMLv2Prof]	S. Cantor et al., " <i>Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0</i> ". OASIS SSTC, August 2004. Document ID sstc-saml-profiles-2.0-cd-01, WORK IN PROGRESS. See http://www.oasis-open.org/committees/security/
1615		
1616		
1617		
1618	[SAMLv2Sec]	F. Hirsch et al., " <i>Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0</i> ". OASIS SSTC, August 2004, WORK IN PROGRESS. Document ID sstc-saml-sec-consider-2.0-cd-01. See http://www.oasis-open.org/committees/security/
1619		
1620		
1621		
1622	[SSTC]	"OASIS Security Services Technical Committee". See http://www.oasis-open.org/committees/security/
1623		
1624	[XML]	Bray, T., Paoli, J., Sperberg-McQueen, C.M. and E. Maler, François Yergeau, " <i>Extensible Markup Language (XML) 1.0 (Third Edition)</i> ", World Wide Web Consortium Recommendation REC-xml, Feb 2004, Available as http://www.w3.org/TR/REC-xml/
1625		
1626		
1627		

1628 Appendix A. Acknowledgments

1629 The editors would like to acknowledge the contributions of the OASIS Security Services Technical
1630 Committee, whose voting members at the time of publication were:

- 1631 • Conor Cahill, AOL
- 1632 • John Hughes, ATOS Origin
- 1633 • Hal Lockhart, BEA Systems
- 1634 • Rick Randall, Booz Allen Hamilton
- 1635 • Ronald Jacobson, Computer Associates
- 1636 • Gavenraj Sodhi, Computer Associates
- 1637 • Tim Alsop, CyberSafe Limited
- 1638 • Paul Madsen, Entrust
- 1639 • Carolina Canales-Valenzuela, Ericsson
- 1640 • Dana Kaufman, Forum Systems
- 1641 • Irving Reid, Hewlett-Packard
- 1642 • Paula Austel, IBM
- 1643 • Maryann Hondo, IBM
- 1644 • Michael McIntosh, IBM
- 1645 • Anthony Nadalin, IBM
- 1646 • Nick Ragouzis, Individual
- 1647 • Scott Cantor, Internet2
- 1648 • Bob Morgan, Internet2
- 1649 • Prateek Mishra, Netegrity
- 1650 • Forest Yin, Netegrity
- 1651 • Peter Davis, Neustar
- 1652 • Frederick Hirsch, Nokia
- 1653 • John Kemp, Nokia
- 1654 • Senthil Sengodan, Nokia
- 1655 • Scott Kiestler, Novell
- 1656 • Cameron Morris, Novell
- 1657 • Charles Knouse, Oblix
- 1658 • Steve Anderson, OpenNetwork
- 1659 • Ari Kermaier, Oracle
- 1660 • Vamsi Motukuru, Oracle
- 1661 • Darren Platt, Ping Identity
- 1662 • Jim Lien, RSA Security
- 1663 • John Linn, RSA Security
- 1664 • Rob Philpott, RSA Security
- 1665 • Dipak Chopra, SAP
- 1666 • Jahan Moreh, Sigaba
- 1667 • Bhavna Bhatnagar, Sun Microsystems
- 1668 • Jeff Hodges, Sun Microsystems
- 1669 • Eve Maler, Sun Microsystems

- 1670 • Ronald Monzillo, Sun Microsystems
- 1671 • Emily Xu, Sun Microsystems
- 1672 • Mike Beach, Boeing
- 1673 • Greg Whitehead, Trustgenix

1674 •
1674 The editors also would like to acknowledge the following people for their contributions to previous versions
1675 of the OASIS Security Assertions Markup Language Standard:

- 1676 • Stephen Farrell, Baltimore Technologies
- 1677 • David Orchard, BEA Systems
- 1678 • Krishna Sankar, Cisco Systems
- 1679 • Zahid Ahmed, CommerceOne
- 1680 • Carlisle Adams, Entrust
- 1681 • Tim Moses, Entrust
- 1682 • Nigel Edwards, Hewlett-Packard
- 1683 • Joe Pato, Hewlett-Packard
- 1684 • Bob Blakley, IBM
- 1685 • Marlena Erdos, IBM
- 1686 • Marc Chanliau, Netegrity
- 1687 • Chris McLaren, Netegrity
- 1688 • Lynne Rosenthal, NIST
- 1689 • Mark Skall, NIST
- 1690 • Simon Godik, Overxeer
- 1691 • Charles Norwood, SAIC
- 1692 • Evan Prodromou, Securant
- 1693 • Robert Griffin, RSA Security (former editor)
- 1694 • Sai Allarvarpu, Sun Microsystems
- 1695 • Chris Ferris, Sun Microsystems
- 1696 • Emily Xu, Sun Microsystems
- 1697 • Mike Myers, Traceroute Security
- 1698 • Phillip Hallam-Baker, VeriSign (former editor)
- 1699 • James Vanderbeek, Vodafone
- 1700 • Mark O'Neill, Vordel
- 1701 • Tony Palmer, Vordel

1702 Finally, the editors wish to acknowledge the following people for their contributions of material used as
1703 input to the OASIS Security Assertions Markup Language specifications:

- 1704 • Thomas Gross, IBM
- 1705 • Birgit Pfitzmann, IBM

1706 **Appendix B. Notices**

1707 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that
1708 might be claimed to pertain to the implementation or use of the technology described in this document or
1709 the extent to which any license under such rights might or might not be available; neither does it represent
1710 that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to
1711 rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made
1712 available for publication and any assurances of licenses to be made available, or the result of an attempt
1713 made to obtain a general license or permission for the use of such proprietary rights by implementors or
1714 users of this specification, can be obtained from the OASIS Executive Director.

1715 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or
1716 other proprietary rights which may cover technology that may be required to implement this specification.
1717 Please address the information to the OASIS Executive Director.

1718 **Copyright © OASIS Open 2004. All Rights Reserved.**

1719 This document and translations of it may be copied and furnished to others, and derivative works that
1720 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and
1721 distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and
1722 this paragraph are included on all such copies and derivative works. However, this document itself may
1723 not be modified in any way, such as by removing the copyright notice or references to OASIS, except as
1724 needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights
1725 defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it
1726 into languages other than English.

1727 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors
1728 or assigns.

1729 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1730 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
1731 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
1732 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.