
Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0 – Errata Composite

Working Draft, ~~August 2006~~12 February 2007

Document identifier:

sstc-saml-bindings-errata-2.0-wd-044

Location:

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

Editors:

Scott Cantor, Internet2
Frederick Hirsch, Nokia
John Kemp, Nokia
Rob Philpott, RSA Security
~~Jahan Moreh, Sigaba (errata document editor)~~
Eve Maler, Sun Microsystems (errata ~~composite document~~ editor)

Contributors to the Errata:

[Nick Ragouzis, Enosis Group](#)
[Thomas Wisniewski, Entrust](#)
[Greg Whitehead, HP](#)
[Heather Hinton, IBM](#)
[Connor P. Cahill, Intel](#)
[Scott Cantor, Internet2](#)
[Eric Tiffany, Liberty Alliance](#)
[Tom Scavo, NCSA/University of Illinois](#)
[Jeff Hodges, Neustar](#)
[Ari Kermaier, Oracle](#)
[Prateek Mishra, Oracle](#)
[Brian Campbell, Ping Identity](#)
[Jim Lien, RSA Security](#)
[Rob Philpott, RSA Security](#)
[Jahan Moreh, Sigaba](#)
[Emily Xu, Sun Microsystems](#)
[David Staggs, Veteran's Health Administration](#)

SAML V2.0 Contributors:

Conor P. Cahill, AOL
John Hughes, Atos Origin
Hal Lockhart, BEA Systems
Michael Beach, Boeing
Rebekah Metz, Booz Allen Hamilton
Rick Randall, Booz Allen Hamilton
Thomas Wisniewski, Entrust
Irving Reid, Hewlett-Packard
Paula Austel, IBM
Maryann Hondo, IBM
Michael McIntosh, IBM
Tony Nadalin, IBM

48 Nick Ragouzis, Individual
49 Scott Cantor, Internet2
50 RL 'Bob' Morgan, Internet2
51 Peter C Davis, Neustar
52 Jeff Hodges, Neustar
53 Frederick Hirsch, Nokia
54 John Kemp, Nokia
55 Paul Madsen, NTT
56 Steve Anderson, OpenNetwork
57 Prateek Mishra, Principal Identity
58 John Linn, RSA Security
59 Rob Philpott, RSA Security
60 Jahan Moreh, Sigaba
61 Anne Anderson, Sun Microsystems
62 Eve Maler, Sun Microsystems
63 Ron Monzillo, Sun Microsystems
64 Greg Whitehead, Trustgenix

65 **Abstract:**

66 The SAML V2.0 Bindings specification defines protocol bindings for the use of SAML assertions
67 and request-response messages in communications protocols and frameworks. This document,
68 known as an “errata composite”, combines corrections to reported errata with the original
69 specification text. By design, the corrections are limited to clarifications of ambiguous or
70 conflicting specification text. This document shows deletions from the original specification as
71 struck-through text, and additions as ~~colored~~blue underlined text. The “[PE~~nn~~” designations
72 embedded in the text refer to particular errata and their dispositions.

73 **Status:**

74 This errata composite document is a **working draft** based on the [original](#) OASIS Standard
75 document that had been produced by the Security Services Technical Committee and approved
76 by the OASIS membership on 1 March 2005. While the errata corrections appearing here are
77 non-normative, they reflect ~~the consensus of the TC about how to interpret the specification and~~
78 ~~are likely to be incorporated into any future standards-track revision of the SAML-~~
79 ~~specification~~changes specified by the Approved Errata document (currently at Working Draft
80 revision 02), which is on an OASIS standardization track. In case of any discrepancy between this
81 document and the Approved Errata, the latter has precedence. See also the Errata Working
82 Document (currently at revision 39), which provides background on the changes specified here.

83 This document includes ~~errata~~ corrections ~~for errata through revision 33 of the errata document,~~
84 ~~including PE1, PE2, PE4, PE19, PE21, PE24, PE31, P and E57, and E59.~~

85 Committee members should submit comments and potential errata to the [security-](mailto:security-services@lists.oasis-open.org)
86 [services@lists.oasis-open.org](mailto:security-services@lists.oasis-open.org) list. Others should submit them by following the instructions at ~~at-~~
87 http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=security.

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

Table of Contents

93	1 Introduction.....	5
94	1.1 Protocol Binding Concepts.....	5
95	1.2 Notation.....	5
96	2 Guidelines for Specifying Additional Protocol Bindings.....	7
97	3 Protocol Bindings.....	8
98	3.1 General Considerations.....	8
99	3.1.1 Use of RelayState.....	8
100	3.1.2 Security.....	8
101	3.1.2.1 Use of SSL 3.0 or TLS 1.0.....	8
102	3.1.2.2 Data Origin Authentication.....	8
103	3.1.2.3 Message Integrity.....	8
104	3.1.2.4 Message Confidentiality.....	9
105	3.1.2.5 Security Considerations.....	9
106	3.2 SAML SOAP Binding.....	9
107	3.2.1 Required Information.....	9
108	3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding.....	10
109	3.2.2.1 Basic Operation.....	10
110	3.2.2.2 SOAP Headers.....	10
111	3.2.3 Use of SOAP over HTTP.....	11
112	3.2.3.1 HTTP Headers.....	11
113	3.2.3.2 Caching.....	11
114	3.2.3.3 Error Reporting.....	11
115	3.2.3.4 Metadata Considerations.....	12
116	3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP.....	12
117	3.3 Reverse SOAP (PAOS) Binding.....	13
118	3.3.1 Required Information.....	13
119	3.3.2 Overview.....	13
120	3.3.3 Message Exchange.....	13
121	3.3.3.1 HTTP Request, SAML Request in SOAP Response.....	14
122	3.3.3.2 SAML Response in SOAP Request, HTTP Response.....	15
123	3.3.4 Caching.....	15
124	3.3.5 Security Considerations.....	15
125	3.3.5.1 Error Reporting.....	15
126	3.3.5.2 Metadata Considerations.....	15
127	3.4 HTTP Redirect Binding.....	15
128	3.4.1 Required Information.....	16
129	3.4.2 Overview.....	16
130	3.4.3 RelayState.....	16
131	3.4.4 Message Encoding.....	16
132	3.4.4.1 DEFLATE Encoding.....	17
133	3.4.5 Message Exchange.....	18
134	3.4.5.1 HTTP and Caching Considerations.....	19
135	3.4.5.2 Security Considerations.....	19
136	3.4.6 Error Reporting.....	20
137	3.4.7 Metadata Considerations.....	20
138	3.4.8 Example SAML Message Exchange Using HTTP Redirect.....	20

139	3.5 HTTP POST Binding.....	21
140	3.5.1 Required Information.....	21
141	3.5.2 Overview.....	21
142	3.5.3 RelayState.....	22
143	3.5.4 Message Encoding.....	22
144	3.5.5 Message Exchange.....	22
145	3.5.5.1 HTTP and Caching Considerations.....	23
146	3.5.5.2 Security Considerations.....	24
147	3.5.6 Error Reporting.....	24
148	3.5.7 Metadata Considerations.....	24
149	3.5.8 Example SAML Message Exchange Using HTTP POST.....	24
150	3.6 HTTP Artifact Binding.....	26
151	3.6.1 Required Information.....	27
152	3.6.2 Overview.....	27
153	3.6.3 Message Encoding.....	27
154	3.6.3.1 RelayState.....	27
155	3.6.3.2 URL Encoding.....	27
156	3.6.3.3 Form Encoding.....	28
157	3.6.4 Artifact Format.....	28
158	3.6.4.1 Required Information.....	29
159	3.6.4.2 Format Details.....	29
160	3.6.5 Message Exchange.....	29
161	3.6.5.1 HTTP and Caching Considerations.....	31
162	3.6.5.2 Security Considerations.....	31
163	3.6.6 Error Reporting.....	32
164	3.6.7 Metadata Considerations.....	32
165	3.6.8 Example SAML Message Exchange Using HTTP Artifact.....	32
166	3.7 SAML URI Binding.....	35
167	3.7.1 Required Information.....	35
168	3.7.2 Protocol-Independent Aspects of the SAML URI Binding.....	35
169	3.7.2.1 Basic Operation.....	35
170	3.7.3 Security Considerations.....	36
171	3.7.4 MIME Encapsulation.....	36
172	3.7.5 Use of HTTP URIs.....	36
173	3.7.5.1 URI Syntax.....	36
174	3.7.5.2 HTTP and Caching Considerations.....	36
175	3.7.5.3 Security Considerations.....	36
176	3.7.5.4 Error Reporting.....	37
177	3.7.5.5 Metadata Considerations.....	37
178	3.7.5.6 Example SAML Message Exchange Using an HTTP URI.....	37
179	4 References.....	38
180	Appendix A. Registration of MIME media type application/samlassertion+xml.....	40
181	Appendix B. Acknowledgments.....	44
182	Appendix C. Notices.....	46

1 Introduction

183

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

186 The SAML assertions and protocols specification [SAMLCore] defines the SAML assertions and request-
187 response messages themselves, and the SAML profiles specification [SAMLProfile] defines specific
188 usage patterns that reference both [SAMLCore] and bindings defined in this specification or elsewhere.
189 The SAML conformance document [SAMLConform] lists all of the specifications that comprise SAML
190 V2.0.

1.1 Protocol Binding Concepts

191

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

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

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

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

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

1.2 Notation

206

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

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

211 `Example code listings appear like this.`

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

213 Conventional XML namespace prefixes are used throughout this specification to stand for their respective
214 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

Prefix	XML Namespace	Comments
		Syntax and Processing specification [XMLSig] and its governing schema.
SOAP-ENV:	http://schemas.xmlsoap.org/soap/envelope	This namespace is defined in SOAP V1.1 [SOAP11].

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

2 Guidelines for Specifying Additional Protocol Bindings

218
219

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

225 The SSTC welcomes submission of proposals from OASIS members for new protocol bindings. OASIS
226 members may wish to submit these proposals for consideration by the SSTC in a future version of this
227 specification. Other members may simply wish to inform the committee of their work related to SAML.
228 Please refer to the SSTC web site [SSTCWeb] for further details on how to submit such proposals to the
229 SSTC.

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

- 231 1. Specify three pieces of identifying information: a URI that uniquely identifies the protocol binding,
232 postal or electronic contact information for the author, and a reference to previously defined
233 bindings or profiles that the new binding updates or obsoletes.
- 234 2. Describe the set of interactions between parties involved in the binding. Any restrictions on
235 applications used by each party and the protocols involved in each interaction must be explicitly
236 called out.
- 237 3. Identify the parties involved in each interaction, including how many parties are involved and
238 whether intermediaries may be involved.
- 239 4. Specify the method of authentication of parties involved in each interaction, including whether
240 authentication is required and acceptable authentication types.
- 241 5. Identify the level of support for message integrity, including the mechanisms used to ensure
242 message integrity.
- 243 6. Identify the level of support for confidentiality, including whether a third party may view the contents
244 of SAML messages and assertions, whether the binding requires confidentiality, and the
245 mechanisms recommended for achieving confidentiality.
- 246 7. Identify the error states, including the error states at each participant, especially those that receive
247 and process SAML assertions or messages.
- 248 8. Identify security considerations, including analysis of threats and description of countermeasures.
- 249 9. Identify metadata considerations, such that support for a binding involving a particular
250 communications protocol or used in a particular profile can be advertised in an efficient and
251 interoperable way.

252 **3 Protocol Bindings**

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

254 **3.1 General Considerations**

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

256 **3.1.1 Use of RelayState**

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

264 **3.1.2 Security**

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

267 **3.1.2.1 Use of SSL 3.0 or TLS 1.0**

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

272 **3.1.2.2 Data Origin Authentication**

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

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

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

282 **3.1.2.3 Message Integrity**

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

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

288 3.1.2.4 Message Confidentiality

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

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

294 3.1.2.5 Security Considerations

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

299 IETF RFC 2617 [RFC2617] describes possible attacks in the HTTP environment when basic or message-
300 digest authentication schemes are used.

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

302 3.2 SAML SOAP Binding

303 SOAP is a lightweight protocol intended for exchanging structured information in a decentralized,
304 distributed environment [SOAP11]. It uses XML technologies to define an extensible messaging
305 framework providing a message construct that can be exchanged over a variety of underlying protocols.
306 The framework has been designed to be independent of any particular programming model and other
307 implementation specific semantics. Two major design goals for SOAP are simplicity and extensibility.
308 SOAP attempts to meet these goals by omitting, from the messaging framework, features that are often
309 found in distributed systems. Such features include but are not limited to "reliability", "security",
310 "correlation", "routing", and "Message Exchange Patterns" (MEPs).

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

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

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

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

321 3.2.1 Required Information

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

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

324 **Description:** Given below.

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

326 3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding

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

330 3.2.2.1 Basic Operation

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

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

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

337 1. A system entity acting as a SAML requester transmits a SAML request element within the body of
338 a SOAP message to a system entity acting as a SAML responder. The SAML requester MUST
339 NOT include more than one SAML request per SOAP message or include any additional XML
340 elements in the SOAP body.

341 2. The SAML responder ~~[E19]SHOULD~~MUST return a SOAP message containing either a SAML
342 response element in the body or a SOAP fault~~either a SAML response element within the body of~~
343 ~~another SOAP message or generate a SOAP fault~~. The SAML responder MUST NOT include
344 more than one SAML response per SOAP message or include any additional XML elements in the
345 SOAP body. ~~If a SAML responder cannot, for some reason, process a SAML request, it MUST~~
346 ~~generate a SOAP fault~~. SOAP fault codes SHOULD~~MUST~~ NOT be sent for errors within the SAML
347 problem domain, for example, inability to find an extension schema or as a signal that the subject
348 is not authorized to access a resource in an authorization query. See Section 3.2.3.3 for more
349 information about error handling. (SOAP 1.1 faults and fault codes are discussed in [SOAP11]
350 Section 4.1.)

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

355 [SOAP11] references an early draft of the XML Schema specification including an obsolete namespace.
356 SAML requesters SHOULD generate SOAP documents referencing only the final XML schema
357 namespace. SAML responders MUST be able to process both the XML schema namespace used in
358 [SOAP11] as well as the final XML schema namespace.

359 3.2.2.2 SOAP Headers

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

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

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

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

371 3.2.3 Use of SOAP over HTTP

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

375 The HTTP binding for SOAP is described in [SOAP11] Section 6.0. It requires the use of a `SOAPAction`
376 header as part of a SOAP HTTP request. A SAML responder MUST NOT depend on the value of this
377 header. A SAML requester MAY set the value of the `SOAPAction` header as follows:

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

379 3.2.3.1 HTTP Headers

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

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

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

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

391 3.2.3.2 Caching

392 HTTP proxies should not cache SAML protocol messages. To ensure this, the following rules SHOULD be
393 followed.

394 When using HTTP 1.1 [RFC2616], requesters SHOULD:

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

397 When using HTTP 1.1, responders SHOULD:

- 398 • Include a `Cache-Control` header field set to "no-cache, no-store, must-revalidate,
399 private".
- 400 • Include a `Pragma` header field set to "no-cache".
- 401 • NOT include a `Validator`, such as a `Last-Modified` or `ETag` header.

402 3.2.3.3 Error Reporting

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

405 As described in [SOAP11] Section 6.2, in the case of a SOAP error while processing a SOAP request, the
406 SOAP HTTP server MUST return a "500 Internal Server Error" response and include a SOAP
407 message in the response with a SOAP `<SOAP-ENV:fault>` element. This type of error SHOULD be
408 returned for SOAP-related errors detected before control is passed to the SAML processor, or when the
409 SOAP processor reports an internal error (for example, the SOAP XML namespace is incorrect, the SAML

410 schema cannot be located, the SAML processor throws an exception, and so on).

411 In the case of a SAML processing error, the SOAP HTTP server **[E19]SHOULDMUST** respond with "200
412 OK" and include a SAML-specified `<samlp:Status>` element in the SAML response within the SOAP
413 body. Note that the `<samlp:Status>` element does not appear by itself in the SOAP body, but only
414 within a SAML response of some sort.

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

417 3.2.3.4 Metadata Considerations

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

421 3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP

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

```
424 POST /SamlService HTTP/1.1
425 Host: www.example.com
426 Content-Type: text/xml
427 Content-Length: nnn
428 SOAPAction: http://www.oasis-open.org/committees/security
429 <SOAP-ENV:Envelope
430   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
431   <SOAP-ENV:Body>
432     <samlp:AttributeQuery xmlns:samlp="..."
433     xmlns:saml="..." xmlns:ds="..." ID="_6c3a4f8b9c2d" Version="2.0"
434     IssueInstant="2004-03-27T08:41:00Z"
435       <ds:Signature> ... </ds:Signature>
436       <saml:Subject>
437         ...
438       </saml:Subject>
439     </samlp:AttributeQuery>
440   </SOAP-ENV:Body>
441 </SOAP-ENV:Envelope>
```

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

```
444 HTTP/1.1 200 OK
445 Content-Type: text/xml
446 Content-Length: nnnn
447 <SOAP-ENV:Envelope
448   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
449   <SOAP-ENV:Body>
450     <samlp:Response xmlns:samlp="..." xmlns:saml="..." xmlns:ds="..."
451     ID="_6c3a4f8b9c2d" Version="2.0" IssueInstant="2004-03-27T08:42:00Z">
452       <saml:Issuer>https://www.example.com/SAML</saml:Issuer>
453       <ds:Signature> ... </ds:Signature>
454       <Status>
455         <StatusCode Value="..." />
456       </Status>
457
458       <saml:Assertion>
459         <saml:Subject>
460           ...
461         </saml:Subject>
462         <saml:AttributeStatement>
463           ...
464         </saml:AttributeStatement>
465       </saml:Assertion>
```

```
466     </samlp:Response>
467     </SOAP-Env:Body>
468 </SOAP-ENV:Envelope>
```

469 3.3 Reverse SOAP (PAOS) Binding

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

473 3.3.1 Required Information

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

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

476 **Description:** Given below.

477 **Updates:** None.

478 3.3.2 Overview

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

486 3.3.3 Message Exchange

487 The PAOS binding includes two component message exchange patterns:

- 488 1. The HTTP requester sends an HTTP request to a SAML requester. The SAML requester responds
489 with an HTTP response containing a SOAP envelope containing a SAML request message.
- 490 2. Subsequently, the HTTP requester sends an HTTP request to the original SAML requester
491 containing a SOAP envelope containing a SAML response message. The SAML requester
492 responds with an HTTP response, possibly in response to the original service request in step 1.

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

- 495 1. The client requests a service using an HTTP request.
- 496 2. The service provider responds with a SAML authentication request. This is sent using a SOAP
497 request, carried in the HTTP response.
- 498 3. The client returns a SOAP response carrying a SAML authentication response. This is sent using a
499 new HTTP request.
- 500 4. Assuming the service provider authentication and authorization is successful, the service provider
501 may respond to the original service request in the HTTP response.

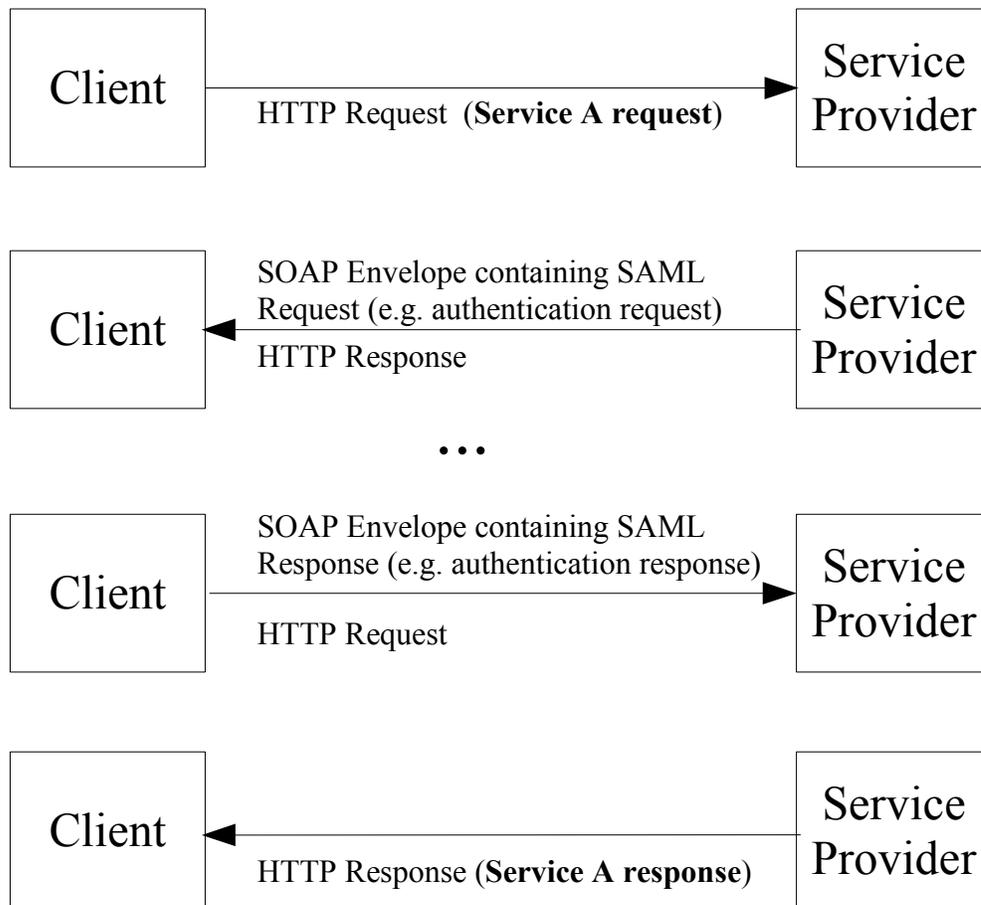


Figure 1: PAOS Binding Message Exchanges

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

- 504 • The HTTP `Accept` Header field MUST indicate an ability to accept the
 505 “application/vnd.paos+xml” content type.
- 506 • The HTTP `PAOS` Header field MUST be present and specify the PAOS version with
 507 “urn:liberty:paos:2003-08”**[E21]-at-a-minimum.**

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

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

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

514 3.3.3.1 HTTP Request, SAML Request in SOAP Response

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

519 Note that while the SAML request message is delivered to the HTTP requester, the actual intended

520 recipient MAY be another system entity, with the HTTP requester acting as an intermediary, as defined by
521 specific profiles.

522 **3.3.3.2 SAML Response in SOAP Request, HTTP Response**

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

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

530 **3.3.4 Caching**

531 HTTP proxies should not cache SAML protocol messages. To ensure this, the following rules SHOULD be
532 followed.

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

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

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

- 537 • Include a `Cache-Control` header field set to "no-cache, no-store, must-revalidate,
538 private".
- 539 • Include a `Pragma` header field set to "no-cache".
- 540 • NOT include a Validator, such as a `Last-Modified` or `ETag` header.

541 **3.3.5 Security Considerations**

542 The HTTP requester in the PAOS binding may act as a SOAP intermediary and when it does, transport
543 layer security for origin authentication, integrity and confidentiality may not meet end-end security
544 requirements. In this case security at the SOAP message layer is ~~[E31]recommended~~**RECOMMENDED**.

545 **3.3.5.1 Error Reporting**

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

549 **3.3.5.2 Metadata Considerations**

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

553 **3.4 HTTP Redirect Binding**

554 The HTTP Redirect binding defines a mechanism by which SAML protocol messages can be transmitted
555 within URL parameters. Permissible URL length is theoretically infinite, but unpredictably limited in
556 practice. Therefore, specialized encodings are needed to carry XML messages on a URL, and larger or

557 more complex message content can be sent using the HTTP POST or Artifact bindings.
558 This binding MAY be composed with the HTTP POST binding (see Section 3.5) and the HTTP Artifact
559 binding (see Section 3.6) to transmit request and response messages in a single protocol exchange using
560 two different bindings.
561 This binding involves the use of a message encoding. While the definition of this binding includes the
562 definition of one particular message encoding, others MAY be defined and used.

563 3.4.1 Required Information

564 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect
565 **Contact information:** security-services-comment@lists.oasis-open.org
566 **Description:** Given below.
567 **Updates:** None.

568 3.4.2 Overview

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

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

577 3.4.3 RelayState

578 RelayState data MAY be included with a SAML protocol message transmitted with this binding. The value
579 MUST NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the
580 message[E1], ~~either via a digital signature (see Section 3.4.4.1) or by some independent means.~~
581 ~~independent of any other protections that may or may not exist during message transmission. Signing is~~
582 ~~not realistic given the space limitation, but because the value is exposed to third-party tampering, the~~
583 ~~entity SHOULD ensure that the value has not been tampered with by using a checksum, a pseudo-~~
584 ~~random value, or similar means.~~

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

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

592 3.4.4 Message Encoding

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

599 messages or content can or cannot be so encoded.

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

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

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

607 **3.4.4.1 DEFLATE Encoding**

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

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

- 612 1. Any signature on the SAML protocol message, including the `<ds:Signature>` XML element itself,
613 MUST be removed. Note that if the content of the message includes another signature, such as a
614 signed SAML assertion, this embedded signature is not removed. However, the length of such a
615 message after encoding essentially precludes using this mechanism. Thus SAML protocol
616 messages that contain signed content SHOULD NOT be encoded using this mechanism.
- 617 2. The DEFLATE compression mechanism, as specified in [RFC1951] is then applied to the entire
618 remaining XML content of the original SAML protocol message.
- 619 3. The compressed data is subsequently base64-encoded according to the rules specified in IETF
620 RFC 2045 [RFC2045]. Linefeeds or other whitespace MUST be removed from the result.
- 621 4. The base-64 encoded data is then URL-encoded, and added to the URL as a query string
622 parameter which MUST be named `SAMLRequest` (if the message is a SAML request) or
623 `SAMLResponse` (if the message is a SAML response).
- 624 5. If RelayState data is to accompany the SAML protocol message, it MUST be URL-encoded and
625 placed in an additional query string parameter named `RelayState`.
- 626 6. If the original SAML protocol message was signed using an XML digital signature, a new signature
627 covering the encoded data as specified above MUST be attached using the rules stated below.

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

- 631 1. The signature algorithm identifier MUST be included as an additional query string parameter,
632 named `SigAlg`. The value of this parameter MUST be a URI that identifies the algorithm used to
633 sign the URL-encoded SAML protocol message, specified according to [XMLSig] or whatever
634 specification governs the algorithm.
- 635 2. To construct the signature, a string consisting of the concatenation of the `RelayState` (if present),
636 `SigAlg`, and `SAMLRequest` (or `SAMLResponse`) query string parameters (each one URL-
637 encoded) is constructed in one of the following ways (ordered as below):
638 `SAMLRequest=value&RelayState=value&SigAlg=value`
639 `SAMLResponse=value&RelayState=value&SigAlg=value`
- 640 3. The resulting string of bytes is the octet string to be fed into the signature algorithm. Any other
641 content in the original query string is not included and not signed.
- 642 4. The signature value MUST be encoded using the base64 encoding (see RFC 2045 [RFC2045]) with
643 any whitespace removed, and included as a query string parameter named `Signature`. Note that
644 some characters in the base64-encoded signature value may themselves require URL-encoding

645 before being added.

646 5. The following signature algorithms (see [XMLSig]) and their URI representations MUST be
647 supported with this encoding mechanism:

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

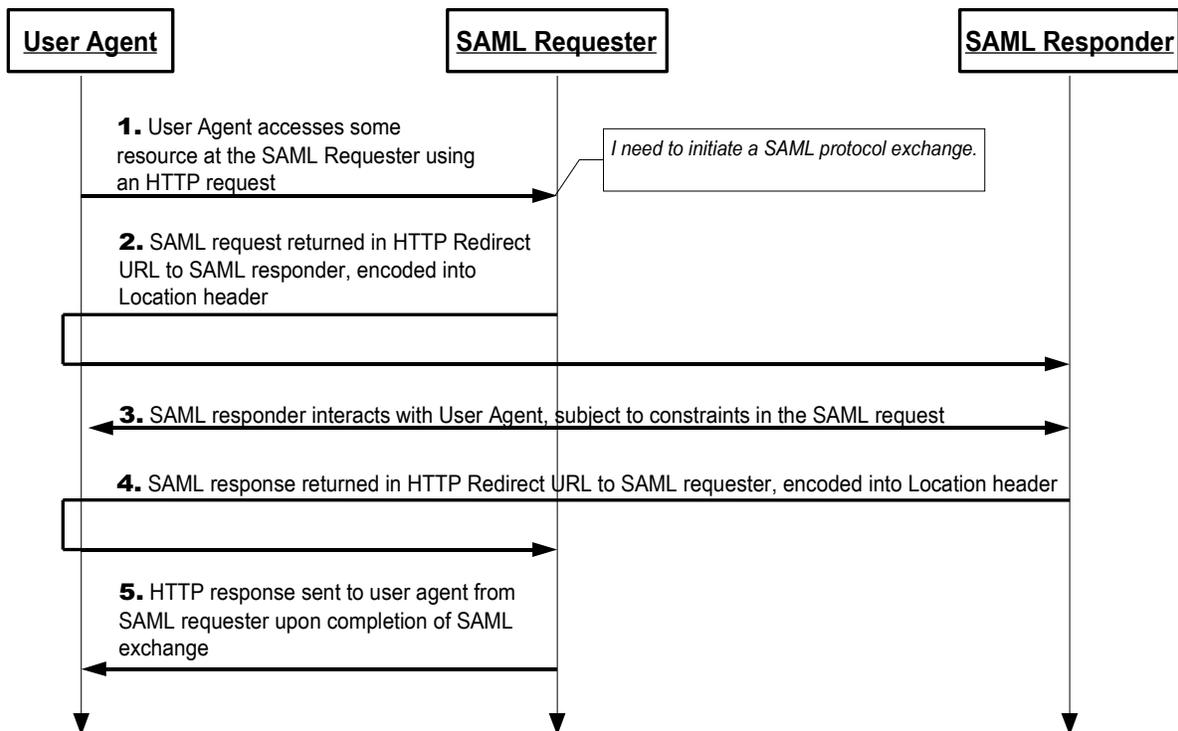
650 Note that when verifying signatures, the order of the query string parameters on the resulting URL to be
651 verified is not prescribed by this binding. The parameters may appear in any order. Before verifying a
652 signature, if any, the relying party MUST ensure that the parameter values to be verified are ordered as
653 required by the signing rules above.

654 Further, note that URL-encoding is not canonical; that is, there are multiple legal encodings for a given
655 value. The relying party MUST therefore perform the verification step using the original URL-encoded
656 values it received on the query string. It is not sufficient to re-encode the parameters after they have been
657 processed by software because the resulting encoding may not match the signer's encoding.

658 Finally, note that if there is no `RelayState` value, the entire parameter should be omitted from the
659 signature computation (and not included as an empty parameter name).

660 3.4.5 Message Exchange

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



666 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
667 processing the request, the system entity decides to initiate a SAML protocol exchange.

668 2. The system entity acting as a SAML requester responds to the HTTP request from the user agent in

669 step 1 by returning a SAML request. The SAML request is returned encoded into the HTTP
670 response's Location header, and the HTTP status MUST be either 303 or 302. The SAML requester
671 MAY include additional presentation and content in the HTTP response to facilitate the user agent's
672 transmission of the message, as defined in HTTP 1.1 [RFC2616]. The user agent delivers the
673 SAML request by issuing an HTTP GET request to the SAML responder.

674 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
675 SAML response or MAY return arbitrary content to facilitate subsequent interaction with the user
676 agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to
677 indicate the requester's level of willingness to permit this kind of interaction (for example, the
678 `IsPassive` attribute in `<samlp:AuthnRequest>`).

679 4. Eventually the responder SHOULD return a SAML response to the user agent to be returned to the
680 SAML requester. The SAML response is returned in the same fashion as described for the SAML
681 request in step 2.

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

684 **3.4.5.1 HTTP and Caching Considerations**

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

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

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

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

691 **3.4.5.2 Security Considerations**

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

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

699 This binding SHOULD NOT be used if the content of the request or response should not be exposed to
700 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is
701 OPTIONAL and depends on the environment of use. If confidentiality is necessary, SSL 3.0 [SSL3] or TLS
702 1.0 [RFC2246] SHOULD be used to protect the message in transit between the user agent and the SAML
703 requester and responder.

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

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

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

712 3.4.6 Error Reporting

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

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

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

720 3.4.7 Metadata Considerations

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

724 3.4.8 Example SAML Message Exchange Using HTTP Redirect

725 In this example, a <LogoutRequest> and <LogoutResponse> message pair is exchanged using the
726 HTTP Redirect binding.

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

```
728 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
729 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
730 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
731 21T19:00:49Z" Version="2.0">  
732 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
733 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
734 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
735 <samlp:SessionIndex>1</samlp:SessionIndex>  
736 </samlp:LogoutRequest>
```

```
737 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
738 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
739 ID="b0730d21b628110d8b7e004005b13a2b"  
740 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"  
741 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">  
742 <Issuer>https://ServiceProvider.com/SAML</Issuer>  
743 <samlp:Status>  
744 <samlp:StatusCode  
745 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>  
746 </samlp:Status>  
747 </samlp:LogoutResponse>
```

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

```
754 HTTP/1.1 302 Object Moved  
755 Date: 21 Jan 2004 07:00:49 GMT
```

```
756 Location:
757 https://ServiceProvider.com/SAML/SLO/Browser?SAMLRequest=fVFdS8MwFH0f7D%2
758 BUvGdNsq62oSsIQyhMESc%2B%2BJYlmRbWpObeyvz3puv2IMjyFM7HPedyK1DdsZdb%2F%2BE
759 HfLFfgwVMtt3RgTwezIEJ72CFqRTnQWJWu7uH7dSLJjsg0ev%2FZFMlttiBWADtt6R%2BSyJ
760 r9msiRH7O70sCm31Mj%2Bo%2BC%2B1KA5G1EWEZaogSQMw2MYBKodrIhjLKONU8FdeSszkVr6
761 T5M0GiHMjvWCknqZXX2OoPxF7kGnaGOuwz%2Fn4L9bY8NC%2By4dulXpRXnxPcXizSZ58KFT
762 eHujEWkNPZylsh9bAMYYUjO2Uiy3jCpTCMo5M1StVjmN9SO150s191U6RV2Dp0vsLIy7NM7YU
763 82r9B90PrvCf85W%2FwL8zSVQzAEAAA%3D%3D&RelayState=0043bfc1bc45110dae170040
764 05b13a2b&SigAlg=http%3A%2F%2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-
765 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE
766 Content-Type: text/html; charset=iso-8859-1
```

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

```
771 HTTP/1.1 302 Object Moved
772 Date: 21 Jan 2004 07:00:49 GMT
773 Location:
774 https://IdentityProvider.com/SAML/SLO/Response?SAMLResponse=fVFNa4QwEL0X%
775 2Bh8k912TaDUGFUp7EbZQ6rKH3mKcbQVNJBOX%2FvxaXQ9tYec0vH1v3nzqkIZ%2BlAf7YSf%
776 2FBjhagxB8Db1BuZQKMjkjrcIOpVEDoPRa1o8vB8n3VI7Oeqtt1bJbbJCB0c7a8j9XTBH9Vy
777 QhqYRbTlrEi4Yo61oUqA0pvShYZHiDQkqs411tAVpeZPqSAGNOkrOas4zzcW55ZlI4liJrTXi
778 BJVBr4wvCJ8777ijbcXZkmaRUxtk7CU7gcB5mLu8pKVddvghd%2Ben9iDIMA3CXTsOrs5euBbf
779 Xdgh%2F9snDK%2FEqW69Ye%2BUnvGL%2F8CfbQnBS%2FQS3z4QLW9aT1oBIws0j%2FG0yAb9%
780 2FV34Dw5k779IBAAA%3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=h
781 ttp%3A%2F%2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-
782 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE
783 Content-Type: text/html; charset=iso-8859-1
```

784 3.5 HTTP POST Binding

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

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

790 3.5.1 Required Information

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

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

793 **Description:** Given below.

794 **Updates:** Effectively replaces the binding aspects of the Browser/POST profile in SAML V1.1
795 [SAML11Bind].

796 3.5.2 Overview

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

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

805 3.5.3 RelayState

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

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

816 If no such [\[E31\]RelayState data](#) value is included with a SAML request message, or if the SAML response
817 message is being generated without a corresponding request, then the SAML responder MAY include
818 RelayState data to be interpreted by the recipient based on the use of a profile or prior agreement
819 between the parties.

820 3.5.4 Message Encoding

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

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

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

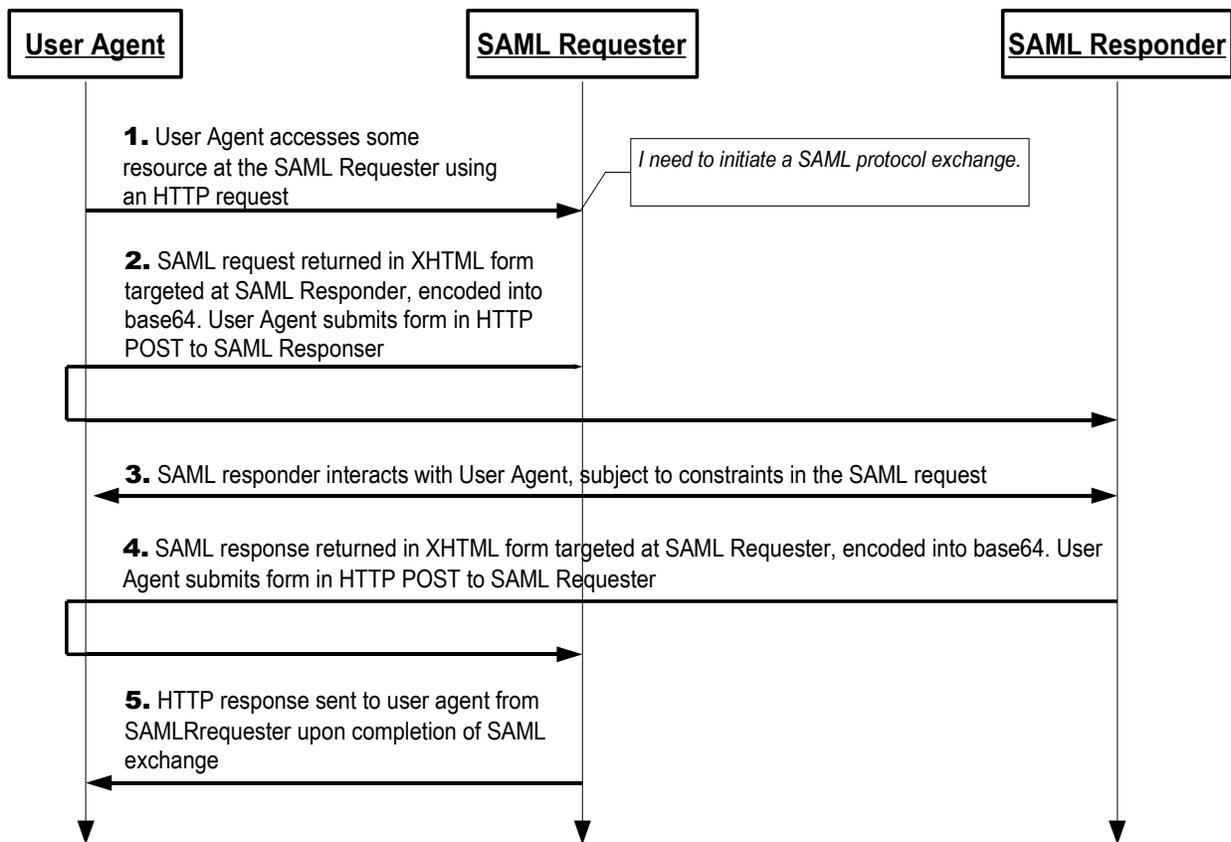
833 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using
834 this binding to which the SAML message is to be delivered. The `method` attribute MUST be "POST".

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

839 Note that any form control values included MUST be transformed so as to be safe to include in the
840 XHTML document. This includes transforming characters such as quotes into HTML entities, etc.

841 3.5.5 Message Exchange

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



- 847 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
848 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 849 2. The system entity acting as a SAML requester responds to an HTTP request from the user agent by
850 returning a SAML request. The request is returned in an XHTML document containing the form and
851 content defined in Section 3.5.4. The user agent delivers the SAML request by issuing an HTTP
852 POST request to the SAML responder.
- 853 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
854 SAML response or it MAY return arbitrary content to facilitate subsequent interaction with the user
855 agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to
856 indicate the requester's level of willingness to permit this kind of interaction (for example, the
857 `IsPassive` attribute in `<samlp:AuthnRequest>`).
- 858 4. Eventually the responder SHOULD return a SAML response to the user agent to be returned to the
859 SAML requester. The SAML response is returned in the same fashion as described for the SAML
860 request in step 2.
- 861 5. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
862 user agent.

863 3.5.5.1 HTTP and Caching Considerations

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

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

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

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

870 **3.5.5.2 Security Considerations**

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

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

879 This binding SHOULD NOT be used if the content of the request or response should not be exposed to
880 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is
881 OPTIONAL and depends on the environment of use. If confidentiality is necessary, SSL 3.0 [SSL3] or TLS
882 1.0 [RFC2246] SHOULD be used to protect the message in transit between the user agent and the SAML
883 requester and responder.

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

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

893 **3.5.6 Error Reporting**

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

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

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

901 **3.5.7 Metadata Considerations**

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

905 **3.5.8 Example SAML Message Exchange Using HTTP POST**

906 In this example, a `<LogoutRequest>` and `<LogoutResponse>` message pair is exchanged using the
907 HTTP POST binding.


```
968 </form>
969 </body>
970 </html>
```

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

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

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

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

988 <form action="https://IdentityProvider.com/SAML/SLO/Response"
989 method="post">
990 <div>
991 <input type="hidden" name="RelayState"
992 value="0043bfclbc45110dae17004005b13a2b"/>
993 <input type="hidden" name="SAMLResponse"
994 value="PHNhbWxwOkxvZ291dFJlc3BvbmlIhHtbG5zOnNhbWxwPSJ1cm46b2FzaXM6bmFt
995 ZXM6dGM6U0FNTDoyLjA6cHJvdG9jb2wiIHhtbG5zPSJ1cm46b2FzaXM6bmFtZXM6
996 dGM6U0FNTDoyLjA6YXNzZXJ0aW9uIgoKICAgIEleEPSJiMdczMGQyMWI2MjgxmTBk
997 OGI3ZTAwNDawNWlzM2EyYiIgcW5SZXNwb25zZVRvPSJkMmI3YzM4OGNlYzY2ZmE3
998 YzY5YzI4ZmQyOTg2NDRhOCINCiAgICBjc3NlZUluZ3RhbnQ9IjIwMDQtMDEtMjFU
999 MTk6MDA6NDlaIiBWXzJzaW9uPSIyLjAiPg0KICAgIDxJc3NlZlZlIHR0cHM6Ly9T
1000 ZXJ2aWNlUHJvdmlkZXIuY29tL1NBTUw8L0lzc3Vlcj4NCiAgICAgICA8c2FtbHA6U3Rh
1001 dHVzPg0KICAgICAgICA8c2FtbHA6U3RhZHVzQ29kZSBWYXl1ZT0idXJuOm9hc2l2
1002 Om5hbWVzOnRjOlNBTUw6Mi4wOnN0YXR1czpTdWNjZXNzIi8+DQogICAgPC9zYW1s
1003 cDpTdGF0dXM+DQo8L3NhbWxwOkxvZ291dFJlc3BvbmlPg==" />
1004 </div>
1005 <noscript>
1006 <div>
1007 <input type="submit" value="Continue"/>
1008 </div>
1009 </noscript>
1010 </form>
1011 </body>
1012 </html>
```

1013 3.6 HTTP Artifact Binding

1014 In the HTTP Artifact binding, the SAML request, the SAML response, or both are transmitted by reference
1015 using a small stand-in called an artifact. A separate, synchronous binding, such as the SAML SOAP
1016 binding, is used to exchange the artifact for the actual protocol message using the artifact resolution
1017 protocol defined in the SAML assertions and protocols specification [SAMLCore].

1018 This binding MAY be composed with the HTTP Redirect binding (see Section 3.4) and the HTTP POST
1019 binding (see Section 3.5) to transmit request and response messages in a single protocol exchange using
1020 two different bindings.

1021 **3.6.1 Required Information**

1022 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact

1023 **Contact information:** security-services-comment@lists.oasis-open.org

1024 **Description:** Given below.

1025 **Updates:** Effectively replaces the binding aspects of the Browser/Artifact profile in SAML V1.1
1026 [SAML11Bind].

1027 **3.6.2 Overview**

1028 The HTTP Artifact binding is intended for cases in which the SAML requester and responder need to
1029 communicate using an HTTP user agent as an intermediary, but the intermediary's limitations preclude or
1030 discourage the transmission of an entire message (or message exchange) through it. This may be for
1031 technical reasons or because of a reluctance to expose the message content to the intermediary (and if
1032 the use of encryption is not practical).

1033 Note that because of the need to subsequently resolve the artifact using another synchronous binding,
1034 such as SOAP, a direct communication path must exist between the SAML message sender and recipient
1035 in the reverse direction of the artifact's transmission (the receiver of the message and artifact must be
1036 able to send a `<samlp:ArtifactResolve>` request back to the artifact issuer). The artifact issuer must
1037 also maintain state while the artifact is pending, which has implications for load-balanced environments.

1038 **3.6.3 Message Encoding**

1039 There are two methods of encoding an artifact for use with this binding. One is to encode the artifact into a
1040 URL parameter and the other is to place the artifact in an HTML form control. When URL encoding is
1041 used, the HTTP GET method is used to deliver the message, while POST is used with form encoding. All
1042 endpoints that support this binding MUST support both techniques.

1043 **3.6.3.1 RelayState**

1044 RelayState data MAY be included with a SAML artifact transmitted with this binding. The value MUST
1045 NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the message
1046 independent of any other protections that may or may not exist during message transmission. Signing is
1047 not realistic given the space limitation, but because the value is exposed to third-party tampering, the
1048 entity SHOULD ensure that the value has not been tampered with by using a checksum, a pseudo-
1049 random value, or similar means.

1050 If an artifact that represents a SAML request is accompanied by RelayState data, then the SAML
1051 responder MUST return its SAML protocol response using a binding that also supports a RelayState
1052 mechanism, and it MUST place the exact data it received with the artifact into the corresponding
1053 RelayState parameter in the response.

1054 If no such value is included with an artifact representing a SAML request, or if the SAML response
1055 message is being generated without a corresponding request, then the SAML responder MAY include
1056 RelayState data to be interpreted by the recipient based on the use of a profile or prior agreement
1057 between the parties.

1058 **3.6.3.2 URL Encoding**

1059 To encode an artifact into a URL, the artifact value is URL-encoded and placed in a query string
1060 parameter named `SAMLart`.

1061 If a "RelayState" value is to accompany the SAML artifact, it MUST be URL-encoded and placed in an

1062 additional query string parameter named `RelayState`.

1063 3.6.3.3 Form Encoding

1064 A SAML artifact is form-encoded by placing it in a hidden form control within a form as defined by
1065 [HTML401], chapter 17. The HTML document MUST adhere to the XHTML specification, [XHTML]. The
1066 form control MUST be named `SAMLart`. Any additional form controls or presentation MAY be included but
1067 MUST NOT be required in order for the recipient to process the artifact.

1068 If a "RelayState" value is to accompany the SAML artifact, it MUST be placed in an additional hidden form
1069 control named `RelayState`, within the same form with the SAML message.

1070 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using
1071 this binding to which the artifact is to be delivered. The `method` attribute MUST be set to "POST".

1072 Any technique supported by the user agent MAY be used to cause the submission of the form, and any
1073 form content necessary to support this MAY be included, such as submit controls and client-side scripting
1074 commands. However, the recipient MUST be able to process the artifact without regard for the
1075 mechanism by which the form submission is initiated.

1076 Note that any form control values included MUST be transformed so as to be safe to include in the
1077 XHTML document. This includes transforming characters such as quotes into HTML entities, etc.

1078 3.6.4 Artifact Format

1079 With respect to this binding, an artifact is a short, opaque string. Different types can be defined and used
1080 without affecting the binding. The important characteristics are the ability of an artifact receiver to identify
1081 the issuer of the artifact, resistance to tampering and forgery, uniqueness, and compactness.

1082 The general format of any artifact includes a mandatory two-byte artifact type code and a two-byte index
1083 value identifying a specific endpoint of the artifact resolution service of the issuer, as follows:

```
1084 SAML_artifact      := B64( TypeCode EndpointIndex RemainingArtifact )
1085 TypeCode           := Byte1Byte2
1086 EndpointIndex     := Byte1Byte2
```

1087 The notation `B64(TypeCode EndpointIndex RemainingArtifact)` stands for the application of
1088 the base64 [RFC2045] transformation to the catenation of the `TypeCode`, `EndpointIndex`, and
1089 `RemainingArtifact`.

1090 The following practices are RECOMMENDED for the creation of SAML artifacts:

- 1091 • Each issuer is assigned an identifying URI, also known as the issuer's entity (or provider) ID. See
1092 Section 8.3.6 of [SAMLCore] for a discussion of this kind of identifier.
- 1093 • The issuer constructs the `SourceID` component of the artifact by taking the SHA-1 hash of the
1094 identification URL. The hash value is NOT encoded into hexadecimal.
- 1095 • The `MessageHandle` value is constructed from a cryptographically strong random or
1096 pseudorandom number sequence [RFC1750] generated by the issuer. The sequence consists of
1097 values of at least 16 bytes in size. These values should be padded as needed to a total length of 20
1098 bytes.

1099 The following describes the single artifact type defined by SAML V2.0. [E4]Although the general artifact
1100 structure resembles that used in prior versions of SAML and the type code of the single format described
1101 below does not conflict with previously defined formats, there is explicitly no correspondence between
1102 SAML V2.0 artifacts and those found in any previous specifications, and artifact formats not defined
1103 specifically for use with SAML V2.0 MUST NOT be used with this binding.

1104 3.6.4.1 Required Information

1105 **Identification:** urn:oasis:names:tc:SAML:2.0:artifact-04

1106 **Contact information:** security-services-comment@lists.oasis-open.org

1107 **Description:** Given below.

1108 **Updates:** None.

1109 3.6.4.2 Format Details

1110 SAML V2.0 defines an artifact type of type code 0x0004. This artifact type is defined as follows:

1111	TypeCode	:= 0x0004
1112	RemainingArtifact	:= SourceID MessageHandle
1113	SourceID	:= 20-byte_sequence
1114	MessageHandle	:= 20-byte_sequence

1115 SourceID is a 20-byte sequence used by the artifact receiver to determine artifact issuer identity and the
1116 set of possible resolution endpoints.

1117 It is assumed that the destination site will maintain a table of SourceID values as well as one or more
1118 indexed URL endpoints (or addresses) for the corresponding SAML responder. The SAML metadata
1119 specification [SAMLMeta] MAY be used for this purpose. On receiving the SAML artifact, the receiver
1120 determines if the SourceID belongs to a known artifact issuer and obtains the location of the SAML
1121 responder using the EndpointIndex before sending a SAML <samlp:ArtifactResolve> message
1122 to it.

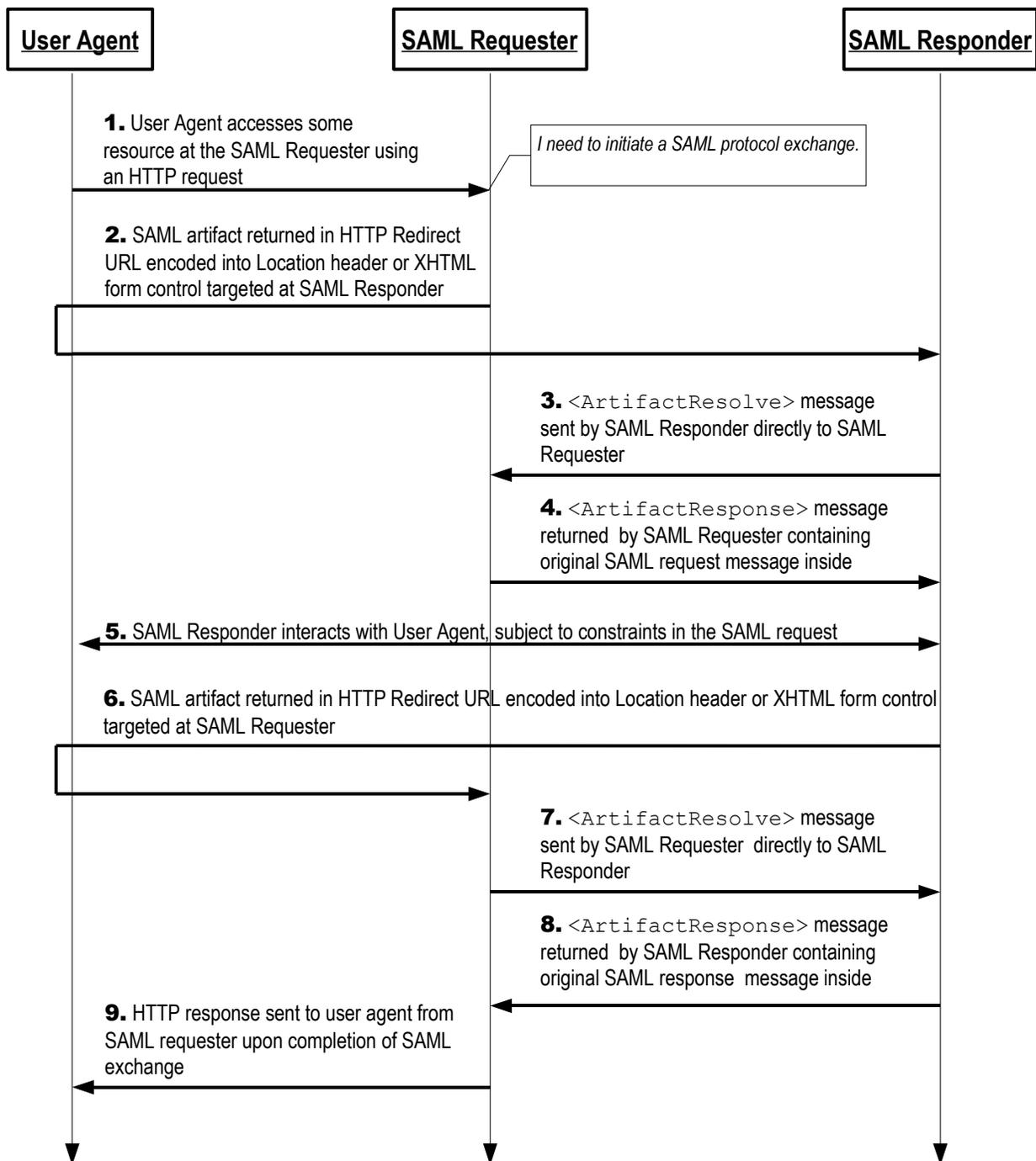
1123 Any two artifact issuers with a common receiver MUST use distinct SourceID values. Construction of
1124 MessageHandle values is governed by the principle that they SHOULD have no predictable relationship
1125 to the contents of the referenced message at the issuing site and it MUST be infeasible to construct or
1126 guess the value of a valid, outstanding message handle.

1127 3.6.5 Message Exchange

1128 The system model used for SAML conversations by means of this binding is a request-response model in
1129 which an artifact reference takes the place of the actual message content, and the artifact reference is
1130 sent to the user agent in an HTTP response and delivered to the message recipient in an HTTP request.
1131 The HTTP interactions before, between, and after these exchanges take place is unspecified. Both the
1132 SAML requester and responder are assumed to be HTTP responders.

1133 Additionally, it is assumed that on receipt of an artifact by way of the user agent, the recipient invokes a
1134 separate, direct exchange with the artifact issuer using the Artifact Resolution Protocol defined in
1135 [SAMLCore]. This exchange MUST use a binding that does not use the HTTP user agent as an
1136 intermediary, such as the SOAP binding. On the successful acquisition of a SAML protocol message, the
1137 artifact is discarded and the processing of the primary SAML protocol exchange resumes (or ends, if the
1138 message is a response).

1139 Issuing and delivering an artifact, along with the subsequent resolution step, constitutes half of the overall
1140 SAML protocol exchange. This binding can be used to deliver either or both halves of a SAML protocol
1141 exchange. A binding composable with it, such as the HTTP Redirect (see Section 3.4) or POST (see
1142 Section 3.5) binding, MAY be used to carry the other half of the exchange. The following sequence
1143 assumes that the artifact binding is used for both halves. See the diagram below illustrating the messages
1144 exchanged.



- 1145 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
 1146 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 1147 2. The system entity acting as a SAML requester responds to an HTTP request from the user agent by
 1148 returning an artifact representing a SAML request.
- 1149 • If URL-encoded, the artifact is returned encoded into the HTTP response's Location
 1150 header, and the HTTP status MUST be either 303 or 302. The SAML requester MAY
 1151 include additional presentation and content in the HTTP response to facilitate the user
 1152 agent's transmission of the message, as defined in HTTP 1.1 [RFC2616]. The user

1153 agent delivers the artifact by issuing an HTTP GET request to the SAML responder.

1154 • If form-encoded, then the artifact is returned in an XHTML document containing the
1155 form and content defined in Section 3.6.3.3. The user agent delivers the artifact by
1156 issuing an HTTP POST request to the SAML responder.

1157 3. The SAML responder determines the SAML requester by examining the artifact (the exact process
1158 depends on the type of artifact), and issues a `<samlp:ArtifactResolve>` request containing
1159 the artifact to the SAML requester using a direct SAML binding, temporarily reversing roles.

1160 4. Assuming the necessary conditions are met, the SAML requester returns a
1161 `<samlp:ArtifactResponse>` containing the original SAML request message it wishes the
1162 SAML responder to process.

1163 5. In general, the SAML responder MAY respond to the SAML request by immediately returning a
1164 SAML artifact or MAY return arbitrary content to facilitate subsequent interaction with the user agent
1165 necessary to fulfill the request. Specific protocols and profiles may include mechanisms to indicate
1166 the requester's level of willingness to permit this kind of interaction (for example, the `IsPassive`
1167 attribute in `<samlp:AuthnRequest>`).

1168 6. Eventually the responder SHOULD return a SAML artifact to the user agent to be returned to the
1169 SAML requester. The SAML response artifact is returned in the same fashion as described for the
1170 SAML request artifact in step 2. The SAML requester determines the SAML responder by examining
1171 the artifact, and issues a `<samlp:ArtifactResolve>` request containing the artifact to the SAML
1172 responder using a [\[E31\]synchronousdirect](#) SAML binding, as in step 3.

1173 7. Assuming the necessary conditions are met, the SAML responder returns a
1174 `<samlp:ArtifactResponse>` containing the SAML response message it wishes the requester to
1175 process, as in step 4.

1176 8. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
1177 user agent.

1178 **3.6.5.1 HTTP and Caching Considerations**

1179 HTTP proxies and the user agent intermediary should not cache SAML artifacts. To ensure this, the
1180 following rules SHOULD be followed.

1181 When returning SAML artifacts using HTTP 1.1, HTTP responders SHOULD:

- 1182 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 1183 • Include a `Pragma` header field set to "no-cache".

1184 There are no other restrictions on the use of HTTP headers.

1185 **3.6.5.2 Security Considerations**

1186 This binding uses a combination of indirect transmission of a message reference followed by a direct
1187 exchange to return the actual message. As a result, the message reference (artifact) need not itself be
1188 authenticated or integrity protected, but the callback request/response exchange that returns the actual
1189 message MAY be mutually authenticated and integrity protected, depending on the environment of use.

1190 If the actual SAML protocol message is intended for a specific recipient, then the artifact's issuer MUST
1191 authenticate the sender of the subsequent `<samlp:ArtifactResolve>` message before returning the
1192 actual message.

1193 The transmission of an artifact to and from the user agent SHOULD be protected with confidentiality; SSL
1194 3.0 [SSL3] or TLS 1.0 [RFC2246] SHOULD be used. The callback request/response exchange that
1195 returns the actual message MAY be protected, depending on the environment of use.

1196 In general, this binding relies on the artifact as a hard-to-forge short-term reference and applies other
1197 security measures to the callback request/response that returns the actual message. All artifacts MUST
1198 have a single-use semantic enforced by the artifact issuer.

1199 Furthermore, it is RECOMMENDED that artifact receivers also enforce a single-use semantic on the
1200 artifact values they receive, to prevent an attacker from interfering with the resolution of an artifact by a
1201 user agent and then resubmitting it to the artifact receiver. If an attempt to resolve an artifact does not
1202 complete successfully, the artifact SHOULD be placed into a blocked artifact list for a period of time that
1203 exceeds a reasonable acceptance period during which the artifact issuer would resolve the artifact.

1204 Note also that there is no mechanism defined to protect the integrity of the relationship between the
1205 artifact and the "RelayState" value, if any. That is, an attacker can potentially recombine a pair of valid
1206 HTTP responses by switching the "RelayState" values associated with each artifact. As a result, the
1207 producer/consumer of "RelayState" information MUST take care not to associate sensitive state
1208 information with the "RelayState" value without taking additional precautions (such as based on the
1209 information in the SAML protocol message retrieved via artifact).

1210 [E59]Finally, note that the use of the Destination attribute in the root SAML element of the protocol
1211 message is unspecified by this binding, because of the message indirection involved.

1212 3.6.6 Error Reporting

1213 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD
1214 return a response message with a second-level <samlp:StatusCode> value of
1215 urn:oasis:names:tc:SAML:2.0:status:RequestDenied.

1216 HTTP interactions during the message exchange MUST NOT use HTTP error status codes to indicate
1217 failures in SAML processing, since the user agent is not a full party to the SAML protocol exchange.

1218 If the issuer of an artifact receives a <samlp:ArtifactResolve> message that it can understand, it
1219 MUST return a <samlp:ArtifactResponse> with a <samlp:StatusCode> value of
1220 urn:oasis:names:tc:SAML:2.0:status:Success, even if it does not return the corresponding
1221 message (for example because the artifact requester is not authorized to receive the message or the
1222 artifact is no longer valid).

1223 For more information about SAML status codes, see the SAML assertions and protocols specification
1224 [SAMLCore].

1225 3.6.7 Metadata Considerations

1226 Support for [E2]receiving messages using the HTTP Artifact binding SHOULD be reflected by indicating
1227 URL endpoints at which requests and responses for a particular protocol or profile should be sent. Either a
1228 single endpoint or distinct request and response endpoints MAY be supplied. One or more indexed
1229 endpoints for processing <samlp:ArtifactResolve> messages SHOULD also be described. Support
1230 for sending messages using this binding SHOULD be accompanied by one or more indexed
1231 <md:ArtifactResolutionService> endpoints for processing <samlp:ArtifactResolve>
1232 messages.

1233 3.6.8 Example SAML Message Exchange Using HTTP Artifact

1234 In this example, a <LogoutRequest> and <LogoutResponse> message pair is exchanged using the
1235 HTTP Artifact binding, with the artifact resolution taking place using the SOAP binding bound to HTTP.

1236 First, here are the actual SAML protocol messages being exchanged:

```
1237 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
1238 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
1239 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
1240 21T19:00:49Z" Version="2.0">
```

```
1241     <Issuer>https://IdentityProvider.com/SAML</Issuer>
1242     <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
1243 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>
1244     <samlp:SessionIndex>1</samlp:SessionIndex>
1245 </samlp:LogoutRequest>
```

```
1246 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1247 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1248 ID="b0730d21b628110d8b7e004005b13a2b"
1249 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"
1250 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">
1251   <Issuer>https://ServiceProvider.com/SAML</Issuer>
1252   <samlp:Status>
1253     <samlp:StatusCode
1254 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1255   </samlp:Status>
1256 </samlp:LogoutResponse>
```

1257 The initial HTTP request from the user agent in step 1 is not defined by this binding. To initiate the logout
1258 protocol exchange, the SAML requester returns the following HTTP response, containing a SAML artifact.
1259 Note that the line feeds in the HTTP Location header below are a result of document formatting, and
1260 there are no line feeds in the actual header value.

```
1261 HTTP/1.1 302 Object Moved
1262 Date: 21 Jan 2004 07:00:49 GMT
1263 Location:
1264 https://ServiceProvider.com/SAML/SLO/Browser?SAMLart=AAQAADWNEw5VT47wcO4z
1265 X%2FiEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU%3D&RelayState=0043bfc1bc45110da
1266 e17004005b13a2b
1267 Content-Type: text/html; charset=iso-8859-1
```

1268 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact
1269 Resolution protocol and the SOAP binding in steps 3 and 4, as follows:

1270 Step 3:

```
1271 POST /SAML/Artifact/Resolve HTTP/1.1
1272 Host: IdentityProvider.com
1273 Content-Type: text/xml
1274 Content-Length: nnn
1275 SOAPAction: http://www.oasis-open.org/committees/security
1276 <SOAP-ENV:Envelope
1277   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
1278   <SOAP-ENV:Body>
1279     <samlp:ArtifactResolve
1280       xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1281       xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1282       ID="_6c3a4f8b9c2d" Version="2.0"
1283       IssueInstant="2004-01-21T19:00:49Z">
1284       <Issuer>https://ServiceProvider.com/SAML</Issuer>
1285       <Artifact>
1286       AAQAADWNEw5VT47wcO4zX/iEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU=
1287       </Artifact>
1288     </samlp:ArtifactResolve>
1289   </SOAP-ENV:Body>
1290 </SOAP-ENV:Envelope>
```

1291 Step 4:

```
1292 HTTP/1.1 200 OK
1293 Date: 21 Jan 2004 07:00:49 GMT
1294 Content-Type: text/xml
1295 Content-Length: nnnn
1296 <SOAP-ENV:Envelope
1297   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
1298   <SOAP-ENV:Body>
1299     <samlp:ArtifactResponse
```

```

1300         xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1301         xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1302         ID="_FQvGknDfws2Z" Version="2.0"
1303         InResponseTo="_6c3a4f8b9c2d"
1304         IssueInstant="2004-01-21T19:00:49Z">
1305         <Issuer>https://IdentityProvider.com/SAML</Issuer>
1306         <samlp:Status>
1307             <samlp:StatusCode
1308                 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1309         </samlp:Status>
1310         <samlp:LogoutRequest ID="d2b7c388cec36fa7c39c28fd298644a8"
1311             IssueInstant="2004-01-21T19:00:49Z"
1312             Version="2.0">
1313             <Issuer>https://IdentityProvider.com/SAML</Issuer>
1314             <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
1315 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>
1316             <samlp:SessionIndex>1</samlp:SessionIndex>
1317         </samlp:LogoutRequest>
1318     </samlp:ArtifactResponse>
1319 </SOAP-ENV:Body>
1320 </SOAP-ENV:Envelope>

```

1321 After any unspecified interactions may have taken place, the SAML responder returns a second SAML
1322 artifact in its HTTP response in step 6:

```

1323 HTTP/1.1 302 Object Moved
1324 Date: 21 Jan 2004 07:05:49 GMT
1325 Location:
1326 https://IdentityProvider.com/SAML/SLO/Response?SAMLart=AAQAAFQIZXv5%2BQaB
1327 aE5qYurHWJO1nAgLAsqfnyidHIggbFU0mlSGFTyQiPc%3D&RelayState=0043bfc1bc45110
1328 dae17004005b13a2b
1329 Content-Type: text/html; charset=iso-8859-1

```

1330 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact
1331 Resolution protocol and the SOAP binding in steps 7 and 8, as follows:

1332 Step 7:

```

1333 POST /SAML/Artifact/Resolve HTTP/1.1
1334 Host: ServiceProvider.com
1335 Content-Type: text/xml
1336 Content-Length: nnn
1337 SOAPAction: http://www.oasis-open.org/committees/security
1338 <SOAP-ENV:Envelope
1339     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1340     <SOAP-ENV:Body>
1341         <samlp:ArtifactResolve
1342             xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1343             xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1344             ID="_ec36fa7c39" Version="2.0"
1345             IssueInstant="2004-01-21T19:05:49Z">
1346             <Issuer>https://IdentityProvider.com/SAML</Issuer>
1347             <Artifact>
1348                 AAQAAFQIZXv5+QaBaE5qYurHWJO1nAgLAsqfnyidHIggbFU0mlSGFTyQiPc=
1349             </Artifact>
1350         </samlp:ArtifactResolve>
1351     </SOAP-ENV:Body>
1352 </SOAP-ENV:Envelope>

```

1353 Step 8:

```

1354 HTTP/1.1 200 OK
1355 Date: 21 Jan 2004 07:05:49 GMT
1356 Content-Type: text/xml
1357 Content-Length: nnnn
1358 <SOAP-ENV:Envelope
1359     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1360     <SOAP-ENV:Body>

```

```

1361     <samlp:ArtifactResponse
1362         xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1363         xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1364         ID="_FQvGknDfws2Z" Version="2.0"
1365         InResponseTo="_ec36fa7c39"
1366         IssueInstant="2004-01-21T19:05:49Z">
1367         <Issuer>https://ServiceProvider.com/SAML</Issuer>
1368         <samlp:Status>
1369             <samlp:StatusCode
1370                 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1371         </samlp:Status>
1372         <samlp:LogoutResponse ID="_b0730d21b628110d8b7e004005b13a2b"
1373             InResponseTo="_d2b7c388cec36fa7c39c28fd298644a8"
1374             IssueInstant="2004-01-21T19:05:49Z"
1375             Version="2.0">
1376             <Issuer>https://ServiceProvider.com/SAML</Issuer>
1377             <samlp:Status>
1378                 <samlp:StatusCode
1379                     Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1380             </samlp:Status>
1381         </samlp:LogoutResponse>
1382     </samlp:ArtifactResponse>
1383 </SOAP-ENV:Body>
1384 </SOAP-ENV:Envelope>

```

1385 3.7 SAML URI Binding

1386 URIs are a protocol-independent means of referring to a resource. This binding is not a general SAML
1387 request/response binding, but rather supports the encapsulation of a `<samlp:AssertionIDRequest>`
1388 message with a single `<saml:AssertionIDRef>` into the resolution of a URI. The result of a successful
1389 request is a SAML `<saml:Assertion>` element (but not a complete SAML response).

1390 Like SOAP, URI resolution can occur over multiple underlying transports. This binding has
1391 [\[E24\]protocoltransport](#)-independent aspects, but also calls out [as mandatory the implementation of HTTP](#)
1392 [URIuse of HTTP with SSL 3.0 \[SSL3\] or TLS 1.0 \[RFC2246\] as REQUIRED \(mandatory to implement\).](#)

1393 3.7.1 Required Information

1394 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:URI

1395 **Contact information:** security-services-comment@lists.oasis-open.org

1396 **Description:** Given below.

1397 **Updates:** None

1398 3.7.2 Protocol-Independent Aspects of the SAML URI Binding

1399 The following sections define aspects of the SAML URI binding that are independent of the underlying
1400 transport protocol of the URI resolution process.

1401 3.7.2.1 Basic Operation

1402 A SAML URI reference identifies a specific SAML assertion. The result of resolving the URI MUST be a
1403 message containing the assertion, or a transport-specific error. The specific format of the message
1404 depends on the underlying transport protocol. If the transport protocol permits the returned content to be
1405 described, such as HTTP 1.1 [RFC2616], then the assertion MAY be encoded in whatever format is
1406 permitted. If not, the assertion MUST be returned in a form which can be unambiguously interpreted as or
1407 transformed into an XML serialization of the assertion.

1408 It MUST be the case that if the same URI reference is resolved in the future, then either the same SAML

1409 assertion, or an error, is returned. That is, the reference MAY be persistent but MUST consistently
1410 reference the same assertion, if any.

1411 **3.7.3 Security Considerations**

1412 Indirect use of a SAML assertion presents dangers if the binding of the reference to the result is not
1413 secure. The particular threats and their severity depend on the use to which the assertion is being put. In
1414 general, the result of resolving a URI reference to a SAML assertion SHOULD only be trusted if the
1415 requester can be certain of the identity of the responder and that the contents have not been modified in
1416 transit.

1417 It is often not sufficient that the assertion itself be signed, because URI references are by their nature
1418 somewhat opaque to the requester. The requester SHOULD have independent means to ensure that the
1419 assertion returned is actually the one that is represented by the URI; this is accomplished by both
1420 authenticating the responder and relying on the integrity of the response.

1421 **3.7.4 MIME Encapsulation**

1422 For resolution protocols that support MIME as a content description and packaging mechanism, the
1423 resulting assertion SHOULD be returned as a MIME entity of type `application/samlassertion+xml`,
1424 as defined by [SAMLmime].

1425 **3.7.5 Use of HTTP URIs**

1426 A SAML authority that claims conformance to the SAML URI binding MUST implement support for HTTP.
1427 This section describes certain specifics of using HTTP URIs, including URI syntax, HTTP headers, and
1428 error reporting.

1429 **3.7.5.1 URI Syntax**

1430 In general, there are no restrictions on the permissible syntax of a SAML URI reference as long as the
1431 SAML authority responsible for the reference creates the message containing it. However, authorities
1432 MUST support a URL endpoint at which an HTTP request can be sent with a single query string
1433 parameter named `ID`. There MUST be no query string in the endpoint URL itself independent of this
1434 parameter.

1435 For example, if the documented endpoint at an authority is "https://saml.example.edu/assertions", a
1436 request for an assertion with an `ID` of `abcde` can be sent to:

1437 `https://saml.example.edu/assertions?ID=abcde`

1438 Note that [\[E31\]the URI syntax does not support](#) the use of wildcards ~~is not allowed for on~~ such `ID`-queries.

1439 **3.7.5.2 HTTP and Caching Considerations**

1440 HTTP proxies MUST NOT cache SAML assertions. To ensure this, the following rules SHOULD be
1441 followed.

1442 When returning SAML assertions using HTTP 1.1, HTTP responders SHOULD:

- 1443 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 1444 • Include a `Pragma` header field set to "no-cache".

1445 **3.7.5.3 Security Considerations**

1446 RFC 2617 [RFC2617] describes possible attacks in the HTTP environment when basic or message-digest
1447 authentication schemes are used.

1448 Use of SSL 3.0 [SSL3] or TLS 1.0 [RFC2246] is STRONGLY RECOMMENDED as a means of
1449 authentication, integrity protection, and confidentiality.

1450 **3.7.5.4 Error Reporting**

1451 As an HTTP protocol exchange, the appropriate HTTP status code SHOULD be used to indicate the result
1452 of a request. For example, a SAML responder that refuses to perform a message exchange with the
1453 SAML requester SHOULD return a "403 Forbidden" response. If the assertion specified is unknown to
1454 the responder, then a "404 Not Found" response SHOULD be returned. In these cases, the content of
1455 the HTTP body is not significant.

1456 **3.7.5.5 Metadata Considerations**

1457 Support for the URI binding over HTTP SHOULD be reflected by indicating a URL endpoint at which
1458 requests for arbitrary assertions are to be sent.

1459 **3.7.5.6 Example SAML Message Exchange Using an HTTP URI**

1460 Following is an example of a request for an assertion.

```
1461 GET /SamlService?ID=abcde HTTP/1.1  
1462 Host: www.example.com
```

1463 Following is an example of the corresponding response, which supplies the requested assertion.

```
1464 HTTP/1.1 200 OK  
1465 Content-Type: application/samlassertion+xml  
1466 Cache-Control: no-cache, no-store  
1467 Pragma: no-cache  
1468 Content-Length: nnnn  
  
1469 <saml:Assertion ID="abcde" ...>  
1470 ...  
1471 </saml:Assertion>
```

4 References

1472

- 1473 **[HTML401]** D. Raggett et al. *HTML 4.01 Specification*. World Wide Web Consortium
1474 Recommendation, December 1999. See <http://www.w3.org/TR/html4>.
- 1475 **[Liberty]** The Liberty Alliance Project. See <http://www.projectliberty.org>.
- 1476 **[PAOS]** R. Aarts. *Liberty Reverse HTTP Binding for SOAP Specification Version*
1477 1.0. Liberty Alliance Project, 2003. See
1478 <https://www.projectliberty.org/specs/liberty-paos-v1.0.pdf>.
- 1479 **[RFC1750]** D. Eastlake et al. *Randomness Recommendations for Security*. IETF
1480 RFC 1750, December 1994. See <http://www.ietf.org/rfc/rfc1750.txt>.
- 1481 **[RFC2045]** N. Freed et al. *Multipurpose Internet Mail Extensions (MIME) Part One:*
1482 *Format of Internet Message Bodies*, IETF RFC 2045, November 1996. See
1483 <http://www.ietf.org/rfc/rfc2045.txt>.
- 1484 **[RFC2119]** S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*.
1485 IETF RFC 2119, March 1997. See <http://www.ietf.org/rfc/rfc2119.txt>.
- 1486 **[RFC2246]** T. Dierks et al. *The TLS Protocol Version 1.0*. IETF RFC 2246, January
1487 1999. See <http://www.ietf.org/rfc/rfc2246.txt>.
- 1488 **[RFC2279]** F. Yergeau. *UTF-8, a transformation format of ISO 10646*. IETF RFC
1489 2279, January 1998. See <http://www.ietf.org/rfc/rfc2279.txt>.
- 1490 **[RFC2616]** R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC
1491 2616, June 1999. See <http://www.ietf.org/rfc/rfc2616.txt>.
- 1492 **[RFC2617]** J. Franks et al. *HTTP Authentication: Basic and Digest Access*
1493 *Authentication*. IETF RFC 2617, June 1999. See
1494 <http://www.ietf.org/rfc/rfc2617.txt>.
- 1495 **[SAML11Bind]** E. Maler et al. *Bindings and Profiles for the OASIS Security Assertion Markup*
1496 *Language (SAML)*. OASIS, September 2003. Document ID oasis-sstc-saml-
1497 bindings-1.1. See <http://www.oasis-open.org/committees/security/>.
- 1498 **[SAMLConform]** P. Mishra et al. *Conformance Requirements for the OASIS Security Assertion*
1499 *Markup Language (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-
1500 conformance-2.0-os. See <http://www.oasis-open.org/committees/security/>.
- 1501 **[SAMLCore]** S. Cantor et al. *Assertions and Protocols for the OASIS Security Assertion*
1502 *Markup Language (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-
1503 core-2.0-os. See <http://www.oasis-open.org/committees/security/>.
- 1504 **[SAMLGloss]** J. Hodges et al. *Glossary for the OASIS Security Assertion Markup Language*
1505 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-glossary-2.0-os.
1506 See <http://www.oasis-open.org/committees/security/>.
- 1507 **[SAMLMeta]** S. Cantor et al. *Metadata for the OASIS Security Assertion Markup Language*
1508 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-metadata-2.0-os.
1509 See <http://www.oasis-open.org/committees/security/>.
- 1510 **[SAMLmime]** ~~[E57]application/saml+xml Media Type Registration, IETF Internet Draft,~~
1511 ~~<http://www.ietf.org/internet-drafts/draft-hodges-saml-mediatype-01.txt>. OASIS~~
1512 ~~Security Services Technical Committee (SSTC). "application/samlassertion+xml~~
1513 ~~MIME Media Type Registration", IANA MIME Media Types Registry~~
1514 ~~[application/samlassertion+xml](http://www.iana.org/assignments/media-types/application/samlassertion+xml), December 2004. See~~
1515 ~~<http://www.iana.org/assignments/media-types/application/samlassertion+xml>.~~
- 1516 **[SAMLProfile]** S. Cantor et al. *Profiles for the OASIS Security Assertion Markup Language*
1517 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-profiles-2.0-os. See
1518 <http://www.oasis-open.org/committees/security/>.

1519	[SAMLSecure]	F. Hirsch et al. <i>Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0</i> . OASIS SSTC, March 2005. Document ID saml-sec-consider-2.0-os. See http://www.oasis-open.org/committees/security/ .
1520		
1521		
1522		
1523	[SOAP11]	D. Box et al. <i>Simple Object Access Protocol (SOAP) 1.1</i> . World Wide Web Consortium Note, May 2000. See http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ .
1524		
1525		
1526	[SOAP-PRIMER]	N. Mitra. <i>SOAP Version 1.2 Part 0: Primer</i> . World Wide Web Consortium Recommendation, June 2003. See http://www.w3.org/TR/soap12-part0/ ,
1527		
1528	[SSL3]	A. Frier et al. <i>The SSL 3.0 Protocol</i> . Netscape Communications Corp, November 1996.
1529		
1530	[SSTCWeb]	OASIS Security Services Technical Committee website, http://www.oasis-open.org/committees/security/ .
1531		
1532	[XHTML]	<i>XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)</i> . World Wide Web Consortium Recommendation, August 2002. See http://www.w3.org/TR/xhtml1/ .
1533		
1534		
1535	[XMLSig]	D. Eastlake et al. <i>XML-Signature Syntax and Processing</i> . World Wide Web Consortium Recommendation, February 2002. See http://www.w3.org/TR/xmlsig-core/ .
1536		
1537		

1538 **Appendix A. Registration of MIME media type**
1539 **application/samlassertion+xml**

1540 **Introduction**

1541 This document defines a MIME media type -- `application/samlassertion+xml` -- for use
1542 with the XML serialization of SAML (Security Assertion Markup Language) assertions.

1543 The SAML specification sets -- [SAMLv1.0], [SAMLv1.1], [SAMLv2.0] -- are work products of the
1544 OASIS Security Services Technical Committee [SSTC]. The SAML specifications define XML-
1545 based constructs with which one may make, and convey, security assertions. Using SAML, one
1546 can assert that an authentication event pertaining to some subject has occurred and convey said
1547 assertion to a relying party, for example.

1548 SAML assertions, which are explicitly versioned, are defined by [SAMLv1Core], [SAMLv11Core],
1549 and [SAMLv2Core].

1550 **MIME media type name**

1551 `application`

1552 **MIME subtype name**

1553 `samlassertion+xml`

1554 **Required parameters**

1555 None

1556 **Optional parameters**

1557 `charset`

1558 Same as `charset` parameter of `application/xml` [RFC3023].

1559 **Encoding considerations**

1560 Same as for `application/xml` [RFC3023].

1561 **Security considerations**

1562 Per their specification, `samlassertion+xml`-typed objects do not contain executable content.
1563 However, SAML assertions are XML-based objects [XML]. As such, they have all of the general
1564 security considerations presented in Section 10 of [RFC3023], as well as additional ones, since
1565 they are explicit security objects. For example, `samlassertion+xml`-typed objects will often
1566 contain data that may identify or pertain to a natural person, and may be used as a basis for
1567 sessions and access control decisions.

1568 To counter potential issues, `samlassertion+xml`-typed objects contain data that should be
1569 signed appropriately by the sender. Any such signature must be verified by the recipient of the
1570 data - both as a valid signature, and as being the signature of the sender. Issuers of
1571 `samlassertion+xml`-typed objects containing SAMLv2 assertions may also encrypt all, or
1572 portions of, the assertions (see [SAMLv2Core]).

1573 In addition, SAML profiles and protocol bindings specify use of secure channels as appropriate.

1574 [SAMLv2.0] incorporates various privacy-protection techniques in its design. For example: opaque
1575 handles, specific to interactions between specific system entities, may be assigned to subjects.
1576 The handles are mappable to wider-context identifiers (e.g. email addresses, account identifiers,
1577 etc) by only the specific parties.

1578 For a more detailed discussion of SAML security considerations and specific security-related
1579 design techniques, please refer to the SAML specifications listed in the below bibliography. The
1580 specifications containing security-specific information have been explicitly listed for each version
1581 of SAML.

1582 **Interoperability considerations**

1583 SAML assertions are explicitly versioned. Relying parties should ensure that they observe
1584 assertion version information and behave accordingly. See chapters on SAML Versioning in
1585 [SAMLv1Core], [SAMLv11Core], or [SAMLv2Core], as appropriate.

1586 **Published specification**

1587 [SAMLv2Bind] explicitly specifies use of the `application/samlassertion+xml` MIME media
1588 type. However, it is conceivable that non-SAMLv2 assertions (i.e., SAMLv1 and/or SAMLv1.1)
1589 might in practice be conveyed using SAMLv2 bindings.

1590 **Applications which use this media type**

1591 Potentially any application implementing SAML, as well as those applications implementing
1592 specifications based on SAML, e.g. those available from the Liberty Alliance [LAP].

1593 **Additional information**

1594 **Magic number(s)**

1595 In general, the same as for `application/xml` [RFC3023]. In particular, the XML root element of the
1596 returned object will have a namespace-qualified name with:

- 1597 – a local name of: `Assertion`
- 1598 – a namespace URI of: one of the version-specific SAML assertion XML
1599 namespace URIs, as defined by the appropriate version-specific SAML "core"
1600 specification (see bibliography).

1601 With SAMLv2.0 specifically, the root element of the returned object may be either
1602 `<saml:Assertion>` or `<saml:EncryptedAssertion>`, where "saml" represents any XML
1603 namespace prefix that maps to the SAMLv2.0 assertion namespace URI:

1604 `urn:oasis:names:tc:SAML:2.0:assertion`

1605 **File extension(s)**

1606 None

1607 **Macintosh File Type Code(s)**

1608 None

1609 Person & email address to contact for further information

1610 This registration is made on behalf of the OASIS Security Services Technical Committee (SSTC)
1611 Please refer to the SSTC website for current information on committee chairperson(s) and their
1612 contact addresses: <http://www.oasis-open.org/committees/security/>. Committee members should
1613 submit comments and potential errata to the security-services@lists.oasis-open.org list. Others
1614 should submit them by filling out the web form located at [http://www.oasis-
1615 open.org/committees/comments/form.php?wg_abbrev=security](http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=security).

1616 Additionally, the SAML developer community email distribution list, [saml-dev@lists.oasis-
1617 open.org](mailto:saml-dev@lists.oasis-open.org), may be employed to discuss usage of the `application/samlassertion+xml`
1618 MIME media type. The "saml-dev" mailing list is publicly archived here: [http://lists.oasis-
1619 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
1620 subscribe, send a message with the single word "subscribe" in the message body, to: [saml-dev-
1621 request@lists.oasis-open.org](mailto:saml-dev-request@lists.oasis-open.org).

1622 Intended usage

1623 COMMON

1624 Author/Change controller

1625 The SAML specification sets are a work product of the OASIS Security Services Technical
1626 Committee (SSTC). OASIS and the SSTC have change control over the SAML specification sets.

1627 Bibliography

- 1628 [LAP] *"Liberty Alliance Project"*. See <http://www.projectliberty.org/>
- 1629 [OASIS] *"Organization for the Advancement of Structured Information Systems"*.
1630 See <http://www.oasis-open.org/>
- 1631 [RFC3023] M. Murata, S. St.Laurent, D. Kohn, "XML Media Types", IETF Request for
1632 Comments 3023, January 2001. Available as [http://www.rfc-
1633 editor.org/rfc/rfc3023.txt](http://www.rfc-editor.org/rfc/rfc3023.txt)
- 1634 [SAMLv1.0] OASIS Security Services Technical Committee, "Security Assertion
1635 Markup Language (SAML) Version 1.0 Specification Set". OASIS
1636 Standard 200205, November 2002. Available as [http://www.oasis-
1637 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)
- 1638 [SAMLv1Bind] Prateek Mishra et al., "Bindings and Profiles for the OASIS Security
1639 Assertion Markup Language (SAML)", OASIS, November 2002.
1640 Document ID oasis-sstc-saml-bindings-1.0. See [http://www.oasis-
1641 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1642 [SAMLv1Core] Phillip Hallam-Baker et al., "Assertions and Protocol for the OASIS
1643 Security Assertion Markup Language (SAML)", OASIS, November 2002.
1644 Document ID oasis-sstc-saml-core-1.0. See [http://www.oasis-
1645 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1646 [SAMLv1Sec] Chris McLaren et al., "Security Considerations for the OASIS Security
1647 Assertion Markup Language (SAML)", OASIS, November 2002.
1648 Document ID oasis-sstc-saml-sec-consider-1.0. See [http://www.oasis-
1649 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1650 [SAMLv1.1] OASIS Security Services Technical Committee, "Security Assertion
1651 Markup Language (SAML) Version 1.1 Specification Set". OASIS
1652 Standard 200308, August 2003. Available as [http://www.oasis-
1653 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)
- 1654 [SAMLv11Bind] E. Maler et al. "Bindings and Profiles for the OASIS Security Assertion
1655 Markup Language (SAML)". OASIS, September 2003. Document ID

1656		oasis-sstc-saml-bindings-1.1. http://www.oasis-open.org/committees/security/
1657		
1658	[SAMLv11Core]	E. Maler et al. "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)". OASIS, September 2003. Document ID oasis-sstc-saml-core-1.1. http://www.oasis-open.org/committees/security/
1659		
1660		
1661	[SAMLv11Sec]	E. Maler et al. "Security Considerations for the OASIS Security Assertion Markup Language (SAML)". OASIS, September 2003. Document ID oasis-sstc-saml-sec-consider-1.1. http://www.oasis-open.org/committees/security/
1662		
1663		
1664		
1665	[SAMLv2.0]	OASIS Security Services Technical Committee, "Security Assertion Markup Language (SAML) Version 2.0 Specification Set". OASIS Standard, 15-Mar-2005. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip
1666		
1667		
1668		
1669	[SAMLv2Bind]	S. Cantor et al., "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-bindings-2.0-os. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf
1670		
1671		
1672		
1673	[SAMLv2Core]	S. Cantor et al., "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-core-2.0-os. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
1674		
1675		
1676		
1677	[SAMLv2Prof]	S. Cantor et al., "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-profiles-2.0-os. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf
1678		
1679		
1680		
1681	[SAMLv2Sec]	F. Hirsch et al., "Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-sec-consider-2.0-os. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf
1682		
1683		
1684		
1685	[SSTC]	"OASIS Security Services Technical Committee". See http://www.oasis-open.org/committees/security/
1686		
1687	[XML]	Bray, T., Paoli, J., Sperberg-McQueen, C.M. and E. Maler, François Yergeau, "Extensible Markup Language (XML) 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml, Feb 2004, Available as http://www.w3.org/TR/REC-xml/
1688		
1689		
1690		

1691 Appendix B. Acknowledgments

1692 The editors would like to acknowledge the contributions of the OASIS Security Services Technical
1693 Committee, whose voting members at the time of publication were:

- 1694 • Conor Cahill, AOL
- 1695 • John Hughes, Atos Origin
- 1696 • Hal Lockhart, BEA Systems
- 1697 • Mike Beach, Boeing
- 1698 • Rebekah Metz, Booz Allen Hamilton
- 1699 • Rick Randall, Booz Allen Hamilton
- 1700 • Ronald Jacobson, Computer Associates
- 1701 • Gavenraj Sodhi, Computer Associates
- 1702 • Thomas Wisniewski, Entrust
- 1703 • Carolina Canales-Valenzuela, Ericsson
- 1704 • Dana Kaufman, Forum Systems
- 1705 • Irving Reid, Hewlett-Packard
- 1706 • Guy Denton, IBM
- 1707 • Heather Hinton, IBM
- 1708 • Maryann Hondo, IBM
- 1709 • Michael McIntosh, IBM
- 1710 • Anthony Nadalin, IBM
- 1711 • Nick Ragouzis, Individual
- 1712 • Scott Cantor, Internet2
- 1713 • Bob Morgan, Internet2
- 1714 • Peter Davis, Neustar
- 1715 • Jeff Hodges, Neustar
- 1716 • Frederick Hirsch, Nokia
- 1717 • Senthil Sengodan, Nokia
- 1718 • Abbie Barbir, Nortel Networks
- 1719 • Scott Kiester, Novell
- 1720 • Cameron Morris, Novell
- 1721 • Paul Madsen, NTT
- 1722 • Steve Anderson, OpenNetwork
- 1723 • Ari Kermaier, Oracle
- 1724 • Vamsi Motukuru, Oracle
- 1725 • Darren Platt, Ping Identity
- 1726 • Prateek Mishra, Principal Identity
- 1727 • Jim Lien, RSA Security
- 1728 • John Linn, RSA Security
- 1729 • Rob Philpott, RSA Security
- 1730 • Dipak Chopra, SAP
- 1731 • Jahan Moreh, Sigaba
- 1732 • Bhavna Bhatnagar, Sun Microsystems

- 1733 • Eve Maler, Sun Microsystems
- 1734 • Ronald Monzillo, Sun Microsystems
- 1735 • Emily Xu, Sun Microsystems
- 1736 • Greg Whitehead, Trustgenix

1737 The editors also would like to acknowledge the following former SSTC members for their contributions to
1738 this or previous versions of the OASIS Security Assertions Markup Language Standard:

- 1739 • Stephen Farrell, Baltimore Technologies
- 1740 • David Orchard, BEA Systems
- 1741 • Krishna Sankar, Cisco Systems
- 1742 • Zahid Ahmed, CommerceOne
- 1743 • Tim Alsop, CyberSafe Limited
- 1744 • Carlisle Adams, Entrust
- 1745 • Tim Moses, Entrust
- 1746 • Nigel Edwards, Hewlett-Packard
- 1747 • Joe Pato, Hewlett-Packard
- 1748 • Bob Blakley, IBM
- 1749 • Marlena Erdos, IBM
- 1750 • Marc Chanliau, Netegrity
- 1751 • Chris McLaren, Netegrity
- 1752 • Lynne Rosenthal, NIST
- 1753 • Mark Skall, NIST
- 1754 • Charles Knouse, Oblix
- 1755 • Simon Godik, Overxeer
- 1756 • Charles Norwood, SAIC
- 1757 • Evan Prodromou, Securant
- 1758 • Robert Griffin, RSA Security (former editor)
- 1759 • Sai Allarvarpu, Sun Microsystems
- 1760 • Gary Ellison, Sun Microsystems
- 1761 • Chris Ferris, Sun Microsystems
- 1762 • Mike Myers, Traceroute Security
- 1763 • Phillip Hallam-Baker, VeriSign (former editor)
- 1764 • James Vanderbeek, Vodafone
- 1765 • Mark O'Neill, Vordel
- 1766 • Tony Palmer, Vordel

1767 Finally, the editors wish to acknowledge the following people for their contributions of material used as
1768 input to the OASIS Security Assertions Markup Language specifications:

- 1769 • Thomas Gross, IBM
- 1770 • Birgit Pfitzmann, IBM

1771 The editors also would like to gratefully acknowledge Jahan Moreh of Sigaba, who during his tenure on
1772 the SSTC was the primary editor of the errata working document and who made major substantive
1773 contributions to all of the errata materials.

1774 **Appendix C. Notices**

1775 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that
1776 might be claimed to pertain to the implementation or use of the technology described in this document or
1777 the extent to which any license under such rights might or might not be available; neither does it represent
1778 that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to
1779 rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made
1780 available for publication and any assurances of licenses to be made available, or the result of an attempt
1781 made to obtain a general license or permission for the use of such proprietary rights by implementors or
1782 users of this specification, can be obtained from the OASIS Executive Director.

1783 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or
1784 other proprietary rights which may cover technology that may be required to implement this specification.
1785 Please address the information to the OASIS Executive Director.

1786 **Copyright © OASIS Open 2005. All Rights Reserved.**

1787 This document and translations of it may be copied and furnished to others, and derivative works that
1788 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and
1789 distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and
1790 this paragraph are included on all such copies and derivative works. However, this document itself may
1791 not be modified in any way, such as by removing the copyright notice or references to OASIS, except as
1792 needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights
1793 defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it
1794 into languages other than English.

1795 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors
1796 or assigns.

1797 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1798 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
1799 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
1800 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.