



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

Committee Draft 03, 14 December 2004

Document identifier:

sstc-saml-bindings-2.0-cd-03

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
Eve Maler, Sun Microsystems

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
Tim Alsop, CyberSafe Limited
Paul Madsen, Entrust
Irving Reid, Hewlett-Packard
Paula Austel, IBM
Maryann Hondo, IBM
Michael McIntosh, IBM
Tony Nadalin, IBM
Nick Ragouzis, Individual
Scott Cantor, Internet2
RL 'Bob' Morgan, Internet2
Peter C Davis, Neustar
Jeff Hodges, Neustar
Frederick Hirsch, Nokia
John Kemp, Nokia
Charles Knouse, Oblix
Steve Anderson, OpenNetwork
Prateek Mishra, Principal Identity
John Linn, RSA Security
Rob Philpott, RSA Security
Jahan Moreh, Sigaba
Anne Anderson, Sun Microsystems
Gary Ellison, Sun Microsystems

45 Eve Maler, Sun Microsystems
46 Ron Monzillo, Sun Microsystems
47 Greg Whitehead, Trustgenix

48 **Abstract:**

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

51 **Status:**

52 This is a **Committee Draft** approved by the Security Services Technical Committee on 14
53 December 2004.

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

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

63 Table of Contents

64	1 Introduction.....	5
65	1.1 Protocol Binding Concepts.....	5
66	1.2 Notation.....	5
67	2 Guidelines for Specifying Additional Protocol Bindings.....	7
68	3 Protocol Bindings.....	8
69	3.1 General Considerations.....	8
70	3.1.1 Use of RelayState.....	8
71	3.1.2 Security.....	8
72	3.1.2.1 Use of SSL 3.0 or TLS 1.0.....	8
73	3.1.2.2 Data Origin Authentication.....	8
74	3.1.2.3 Message Integrity.....	8
75	3.1.2.4 Message Confidentiality.....	9
76	3.1.2.5 Security Considerations.....	9
77	3.2 SAML SOAP Binding.....	9
78	3.2.1 Required Information.....	9
79	3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding.....	10
80	3.2.2.1 Basic Operation.....	10
81	3.2.2.2 SOAP Headers.....	10
82	3.2.3 Use of SOAP over HTTP.....	11
83	3.2.3.1 HTTP Headers.....	11
84	3.2.3.2 Caching.....	11
85	3.2.3.3 Error Reporting.....	11
86	3.2.3.4 Metadata Considerations.....	12
87	3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP.....	12
88	3.3 Reverse SOAP (PAOS) Binding.....	13
89	3.3.1 Required Information.....	13
90	3.3.2 Overview.....	13
91	3.3.3 Message Exchange.....	13
92	3.3.3.1 HTTP Request, SAML Request in SOAP Response.....	14
93	3.3.3.2 SAML Response in SOAP Request, HTTP Response.....	15
94	3.3.4 Caching.....	15
95	3.3.5 Security Considerations.....	15
96	3.3.5.1 Error Reporting.....	15
97	3.3.5.2 Metadata Considerations.....	15
98	3.4 HTTP Redirect Binding.....	15
99	3.4.1 Required Information.....	16
100	3.4.2 Overview.....	16
101	3.4.3 RelayState.....	16
102	3.4.4 Message Encoding.....	16
103	3.4.4.1 DEFLATE Encoding.....	17
104	3.4.5 Message Exchange.....	18
105	3.4.5.1 HTTP and Caching Considerations.....	19
106	3.4.5.2 Security Considerations.....	19
107	3.4.6 Error Reporting.....	19
108	3.4.7 Metadata Considerations.....	20
109	3.4.8 Example SAML Message Exchange Using HTTP Redirect.....	20

110	3.5 HTTP POST Binding.....	21
111	3.5.1 Required Information.....	21
112	3.5.2 Overview.....	21
113	3.5.3 RelayState.....	21
114	3.5.4 Message Encoding.....	22
115	3.5.5 Message Exchange.....	22
116	3.5.5.1 HTTP and Caching Considerations.....	23
117	3.5.5.2 Security Considerations.....	24
118	3.5.6 Error Reporting.....	24
119	3.5.7 Metadata Considerations.....	24
120	3.5.8 Example SAML Message Exchange Using HTTP POST.....	24
121	3.6 HTTP Artifact Binding.....	26
122	3.6.1 Required Information.....	26
123	3.6.2 Overview.....	27
124	3.6.3 Message Encoding.....	27
125	3.6.3.1 RelayState.....	27
126	3.6.3.2 URL Encoding.....	27
127	3.6.3.3 Form Encoding.....	27
128	3.6.4 Artifact Format.....	28
129	3.6.4.1 Required Information.....	28
130	3.6.4.2 Format Details.....	28
131	3.6.5 Message Exchange.....	29
132	3.6.5.1 HTTP and Caching Considerations.....	31
133	3.6.5.2 Security Considerations.....	31
134	3.6.6 Error Reporting.....	32
135	3.6.7 Metadata Considerations.....	32
136	3.6.8 Example SAML Message Exchange Using HTTP Artifact.....	32
137	3.7 SAML URI Binding.....	35
138	3.7.1 Required Information.....	35
139	3.7.2 Protocol-Independent Aspects of the SAML URI Binding.....	35
140	3.7.2.1 Basic Operation.....	35
141	3.7.3 Security Considerations.....	36
142	3.7.4 MIME Encapsulation.....	36
143	3.7.5 Use of HTTP URIs.....	36
144	3.7.5.1 URI Syntax.....	36
145	3.7.5.2 HTTP and Caching Considerations.....	36
146	3.7.5.3 Security Considerations.....	37
147	3.7.5.4 Error Reporting.....	37
148	3.7.5.5 Metadata Considerations.....	37
149	3.7.5.6 Example SAML Message Exchange Using an HTTP URI.....	37
150	4 References.....	38
151	Appendix A. Registration of MIME media type application/samlassertion+xml.....	40
152	Appendix B. Acknowledgments.....	44
153	Appendix C. Notices.....	46

1 Introduction

154

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

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

1.1 Protocol Binding Concepts

162

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

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

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

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

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

1.2 Notation

177

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

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

182 `Example code listings appear like this.`

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

184 Conventional XML namespace prefixes are used throughout this specification to stand for their respective
185 namespaces as follows, whether or not a namespace declaration is present in the example:

Prefix	XML Namespace	Comments
saml:	urn:oasis:names:tc:SAML:2.0:assertion	This is the SAML V2.0 assertion namespace [SAMLCore].
samlp:	urn:oasis:names:tc:SAML:2.0:protocol	This is the SAML V2.0 protocol namespace [SAMLCore].

Prefix	XML Namespace	Comments
ds:	http://www.w3.org/2000/09/xmldsig#	This namespace is defined in the XML Signature Syntax and Processing specification [XMLSig] and its governing schema.
SOAP-ENV:	http://schemas.xmlsoap.org/soap/envelope	This namespace is defined in SOAP V1.1 [SOAP1.1].

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

2 Guidelines for Specifying Additional Protocol Bindings

189
190

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

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

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

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

222 3 Protocol Bindings

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

224 3.1 General Considerations

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

226 3.1.1 Use of RelayState

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

234 3.1.2 Security

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

237 3.1.2.1 Use of SSL 3.0 or TLS 1.0

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

242 3.1.2.2 Data Origin Authentication

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

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

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

252 3.1.2.3 Message Integrity

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

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

258 **3.1.2.4 Message Confidentiality**

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

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

264 **3.1.2.5 Security Considerations**

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

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

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

272 **3.2 SAML SOAP Binding**

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

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

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

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

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

291 **3.2.1 Required Information**

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

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

294 **Description:** Given below.

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

296 3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding

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

300 3.2.2.1 Basic Operation

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

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

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

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

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

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

327 3.2.2.2 SOAP Headers

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

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

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

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

339 3.2.3 Use of SOAP over HTTP

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

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

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

347 3.2.3.1 HTTP Headers

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

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

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

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

359 3.2.3.2 Caching

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

362 When using HTTP 1.1, requesters SHOULD:

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

365 When using HTTP 1.1, responders SHOULD:

- 366 • Include a `Cache-Control` header field set to "no-cache, no-store, must-revalidate,
367 private".
- 368 • Include a `Pragma` header field set to "no-cache".
- 369 • NOT include a `Validator`, such as a `Last-Modified` or `ETag` header.

370 3.2.3.3 Error Reporting

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

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

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

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

385 3.2.3.4 Metadata Considerations

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

389 3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP

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

```
392 POST /SamlService HTTP/1.1
393 Host: www.example.com
394 Content-Type: text/xml
395 Content-Length: nnn
396 SOAPAction: http://www.oasis-open.org/committees/security
397 <SOAP-ENV:Envelope
398   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
399   <SOAP-ENV:Body>
400     <sampl:AttributeQuery xmlns:sampl="..."
401   xmlns:saml="..." xmlns:ds="..." ID="_6c3a4f8b9c2d" Version="2.0"
402   IssueInstant="2004-03-27T08:41:00Z"
403     <ds:Signature> ... </ds:Signature>
404     <saml:Subject>
405       ...
406     </saml:Subject>
407   </sampl:AttributeQuery>
408 </SOAP-ENV:Body>
409 </SOAP-ENV:Envelope>
```

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

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

```
427         <saml:Subject>
428         ...
429         </saml:Subject>
430         <saml:AttributeStatement>
431         ...
432         </saml:AttributeStatement>
433         </saml:Assertion>
434     </samlp:Response>
435 </SOAP-Env:Body>
436 </SOAP-ENV:Envelope>
```

437 3.3 Reverse SOAP (PAOS) Binding

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

441 3.3.1 Required Information

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

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

444 **Description:** Given below.

445 **Updates:** None.

446 3.3.2 Overview

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

454 3.3.3 Message Exchange

455 The PAOS binding includes two component message exchange patterns:

- 456 1. The HTTP requester sends an HTTP request to a SAML requester. The SAML requester responds
457 with an HTTP response containing a SOAP envelope containing a SAML request message.
- 458 2. Subsequently, the HTTP requester sends an HTTP request to the original SAML requester
459 containing a SOAP envelope containing a SAML response message. The SAML requester
460 responds with an HTTP response, possibly in response to the original service request in step 1.

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

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

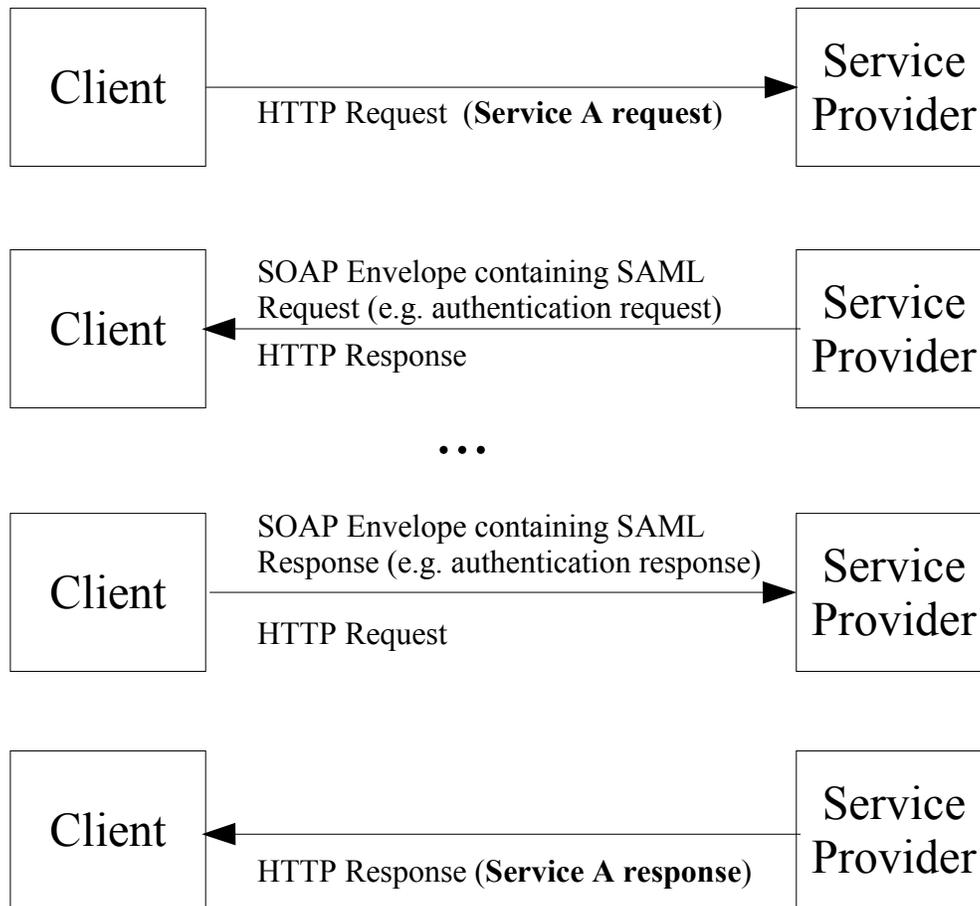


Figure 1: PAOS Binding Message Exchanges

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

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

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

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

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

482 3.3.3.1 HTTP Request, SAML Request in SOAP Response

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

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

490 **3.3.3.2 SAML Response in SOAP Request, HTTP Response**

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

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

498 **3.3.4 Caching**

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

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

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

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

- 505 • Include a `Cache-Control` header field set to "no-cache, no-store, must-revalidate,
506 private".
- 507 • Include a `Pragma` header field set to "no-cache".
- 508 • NOT include a Validator, such as a `Last-Modified` or `ETag` header.

509 **3.3.5 Security Considerations**

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

513 **3.3.5.1 Error Reporting**

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

517 **3.3.5.2 Metadata Considerations**

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

521 **3.4 HTTP Redirect Binding**

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

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

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

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

531 **3.4.1 Required Information**

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

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

534 **Description:** Given below.

535 **Updates:** None.

536 **3.4.2 Overview**

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

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

545 **3.4.3 RelayState**

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

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

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

556 **3.4.4 Message Encoding**

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

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

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

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

571 3.4.4.1 DEFLATE Encoding

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

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

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

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

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

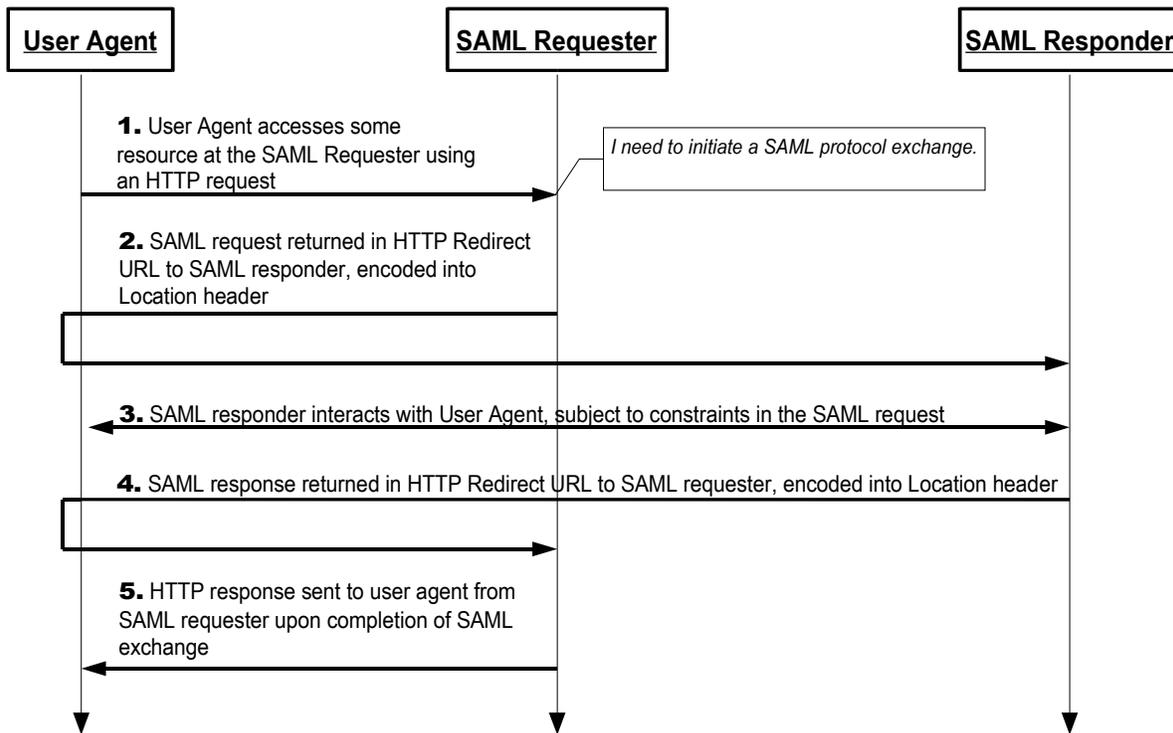
```
602 SAMLRequest=value&RelayState=value&SigAlg=value  
603 SAMLResponse=value&RelayState=value&SigAlg=value
```

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

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

613 3.4.5 Message Exchange

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



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

634 request in step 2.
635 5. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
636 user agent.

637 **3.4.5.1 HTTP and Caching Considerations**

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

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

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

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

644 **3.4.5.2 Security Considerations**

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

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

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

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

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

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

665 **3.4.6 Error Reporting**

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

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

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

673 3.4.7 Metadata Considerations

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

677 3.4.8 Example SAML Message Exchange Using HTTP Redirect

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

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

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

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

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

```
707 HTTP/1.1 302 Object Moved  
708 Date: 21 Jan 2004 07:00:49 GMT  
709 Location:  
710 https://ServiceProvider.com/SAML/SLO/Browser?SAMLRequest=fVFdS8MwFH0f7D%  
711 2BUvGdNsq62oSsIQyhMESc%2B%2BJYlmRbWpObeyvz3puv2IMjyFM7HPedyK1DdsZdb%2F%  
712 2BEHLfFfgvVMt3RgTwzazIEJ72CFqRTnQWJWu7uH7dSLJjsg0ev%2FZFM1ttiBWADtt6R%  
713 2BSyJr9msiRH7070sCm31Mj%2Bo%2BC%  
714 2B1KA5G1EWeZaogSQMw2MYBKodrIhJLkONU8FdeSsZkVr6T5M0GiHMjvWCknqZXZ2OoPx7kG  
715 naGOuwxZ%2Fn4L9bY8NC%  
716 2By4dulXpRXnxPcXizSZ58KfTeHujEWkNPZylsh9bAMYUjO2Uiy3jCpTCMo5M1StVjmn9S01  
717 50s191U6RV2Dp0vsLIy7NM7YU82r9B90PrvCf85W%2FwL8zSVQzAEAAA%3D%  
718 3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F%  
719 2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-  
720 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE  
721 Content-Type: text/html; charset=iso-8859-1
```

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

```
726 HTTP/1.1 302 Object Moved
```

727 Date: 21 Jan 2004 07:00:49 GMT
728 Location:
729 https://IdentityProvider.com/SAML/SLO/Response?SAMLResponse=fvFNa4QwEL0X%
730 2Bh8k912TaDUGFUp7EbZQ6rKH3mKcbQVNJBOX%2FvxaXQ9tYec0vHlv3nzkqIZ%2BlAf7YSf%
731 2FBjhagxB8Db1BuZQKMjkjrcIOpVEDoPRalo8vB8n3VI7OeqttT1bJbbJCB0c7a8j9XTBH9Vy
732 QhqYRbTlrEi4Yo61oUqA0pvShYZHiDQkqs411tAVpeZPqSAgNOkrOas4zzcW55ZlI4liJrTXi
733 BJVBr4wvCJ8777ijbcXZkmaRUxtk7CU7gcB5mLu8pKVddvghd%
734 2Ben9iDIMa3CXTsOrs5euBbfXdgh%2F9snDK%2FEqW69Ye%2BUnvGL%2F8CfbQnBS%
735 2FQS3z4QLW9aT1oBIws0j%2FG0yAb9%2FV34Dw5k779IBAAA%
736 3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F%
737 2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-
738 sha1&Signature=NOTAREALSIGNATUREBUTHEREALONEWOULDGOHERE
739 Content-Type: text/html; charset=iso-8859-1

740 3.5 HTTP POST Binding

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

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

746 3.5.1 Required Information

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

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

749 **Description:** Given below.

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

751 3.5.2 Overview

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

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

760 3.5.3 RelayState

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

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

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

771 **3.5.4 Message Encoding**

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

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

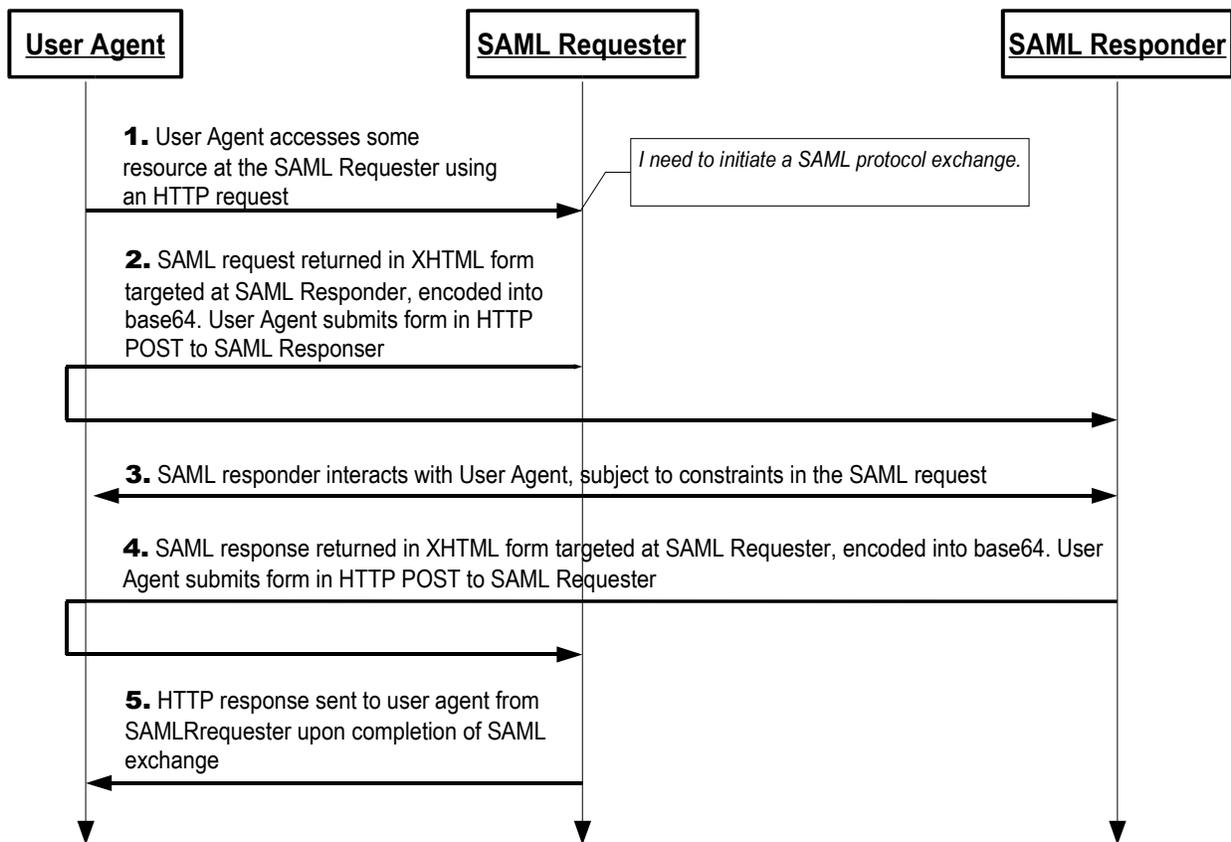
782 If a "RelayState" value is to accompany the SAML protocol message, it MUST be placed in an additional
783 hidden form control named `RelayState` within the same form with the SAML message.

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

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

790 **3.5.5 Message Exchange**

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



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

812 3.5.5.1 HTTP and Caching Considerations

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

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

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

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

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

819 **3.5.5.2 Security Considerations**

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

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

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

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

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

842 **3.5.6 Error Reporting**

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

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

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

850 **3.5.7 Metadata Considerations**

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

854 **3.5.8 Example SAML Message Exchange Using HTTP POST**

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

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

```

858 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
859 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
860 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-
861 21T19:00:49Z" Version="2.0">
862 <Issuer>https://IdentityProvider.com/SAML</Issuer>
863 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
864 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>
865 <samlp:SessionIndex>1</samlp:SessionIndex>
866 </samlp:LogoutRequest>

867 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
868 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
869 ID="b0730d21b628110d8b7e004005b13a2b"
870 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"
871 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">
872 <Issuer>https://ServiceProvider.com/SAML</Issuer>
873 <samlp:Status>
874 <samlp:StatusCode
875 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
876 </samlp:Status>
877 </samlp:LogoutResponse>

```

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

```

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

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

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

895 <form action="https://ServiceProvider.com/SAML/SLO/Browser"
896 method="post">
897 <div>
898 <input type="hidden" name="RelayState"
899 value="0043bfc1bc45110dae17004005b13a2b"/>
900 <input type="hidden" name="SAMLRequest"
901 value="PHNhbWxwOkxvZ291dFJlcXVlc3QeG1sbnM6c2FtbHA9InVybjpvYXNpczpuYW1l
902 czp0YzpqTQU1MOjIuMDpwcm90b2NvbCIgeG1sbnM9InVybjpvYXNpczpuYW1lczp0
903 YzpqTQU1MOjIuMDphc3NlcnRpb24iDQogICAgSUQ9ImQyYjdjMzg4Y2VjMzZmYTdj
904 MzljMjhmZDI5ODY0NGE4IiBjc3N1ZU1uc3RhbnQ9IjIwMDQ0MDUyMjE5MDQ0MDUy
905 NDlaIiBwZXJzaW9uPSIyLjAiPg0KICAgIDxJc3N1ZXI+aHR0cHM6Ly9JZGVudG10
906 eVByb3ZpZGVyLmNvbS9TQU1MPC9Jc3N1ZXI+DQogICAgPE5hbWVJRjRCG3JtYXQ9
907 InVybjpvYXNpczpuYW1lczp0YzpqTQU1MOjIuMDpuYW1laWQtZm90OnBlcnNp
908 c3RlbnQiPjAwNWwNmUwLWFkODItMTEwZC1hNTU2LTAwNDANW1xM2EyYjwvTmFt
909 ZU1EPg0KICAgIDxzYW1scDpTZXNzaW9uSW5kZXg+MTwvc2FtbHA6U2Vzc2l2bkl1
910 ZGV4Pg0KPC9zYW1scDpMb2dvdXR5ZXF1ZXN0Pg==" />
911 </div>
912 <noscript>
913 <div>
914 <input type="submit" value="Continue"/>
915 </div>
916 </noscript>
917 </form>
918 </body>

```

919 </html>

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

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

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

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

937 <form action="https://IdentityProvider.com/SAML/SLO/Response"
938 method="post">
939 <div>
940 <input type="hidden" name="RelayState"
941 value="0043bfclbc45110dae17004005b13a2b"/>
942 <input type="hidden" name="SAMLResponse"
943 value="PHNhbWxwOmxvZ291dFJlc3BvbnNlIHhtbG5zOnNhbWxwPSJ1cm46b2FzaXM6bmFt
944 ZXM6dGM6U0FNTDoyLjA6cHJvdG9jb2wiIHhtbG5zPSJ1cm46b2FzaXM6bmFtZXM6
945 dGM6U0FNTDoyLjA6YXNzZXJ0aW9uIG0KICAgIElEPSJiMdczMGQyMWI2MjgxmTBk
946 OGI3ZTAwNDawNWixM2EyYiIgSW5SZXNwb25zZVRvPSJkMmI3YzZm4OGN1YzZmZmE3
947 YzM5YzI4ZmQyOTg2NDRhOCINCiAgICBjc3NlZUluZ3RhbnQ9IjIwMDQtMDEtMjF1
948 MTk6MDA6NDlaIiBwZXJzaW9uPSIyLjA6cHJvdG9jb2wiIHhtbG5zPSJ1cm46b2FzaXM6bmFt
949 ZXJ2aWNlUHJvdmlkZXIuY29tL1NBTUw8L0lzc3Vlcj4NCiAgICA8c2FtbHA6U3Rh
950 dHVzPg0KICAgICA8c2FtbHA6U3RhZHVzQ29kZSBWYXx1ZT0idXJuOm9hc2lz
951 Om5hbWVzOnRjOlNBTUw6Mi4wOnN0YXR1czpTdWNjZXNzIi8+DQogICA8PC9zYW1s
952 cDpTdGF0dXM+DQo8L3NhbWxwOmxvZ291dFJlc3BvbnNlPg==" />
953 </div>
954 <noscript>
955 <div>
956 <input type="submit" value="Continue"/>
957 </div>
958 </noscript>
959 </form>
960 </body>
961 </html>
```

962 3.6 HTTP Artifact Binding

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

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

970 3.6.1 Required Information

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

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

973 **Description:** Given below.

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

975 **3.6.2 Overview**

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

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

986 **3.6.3 Message Encoding**

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

991 **3.6.3.1 RelayState**

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

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

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

1003 **3.6.3.2 URL Encoding**

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

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

1008 **3.6.3.3 Form Encoding**

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

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

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

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

1021 3.6.4 Artifact Format

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

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

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

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

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

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

1042 The following describes the single artifact type defined by SAML V2.0.

1043 3.6.4.1 Required Information

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

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

1046 **Description:** Given below.

1047 **Updates:** None.

1048 3.6.4.2 Format Details

1049 SAML V2.0 defines an artifact type of type code 0x0004. This artifact type is defined as follows:

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

1053 `MessageHandle` := 20-byte_sequence

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

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

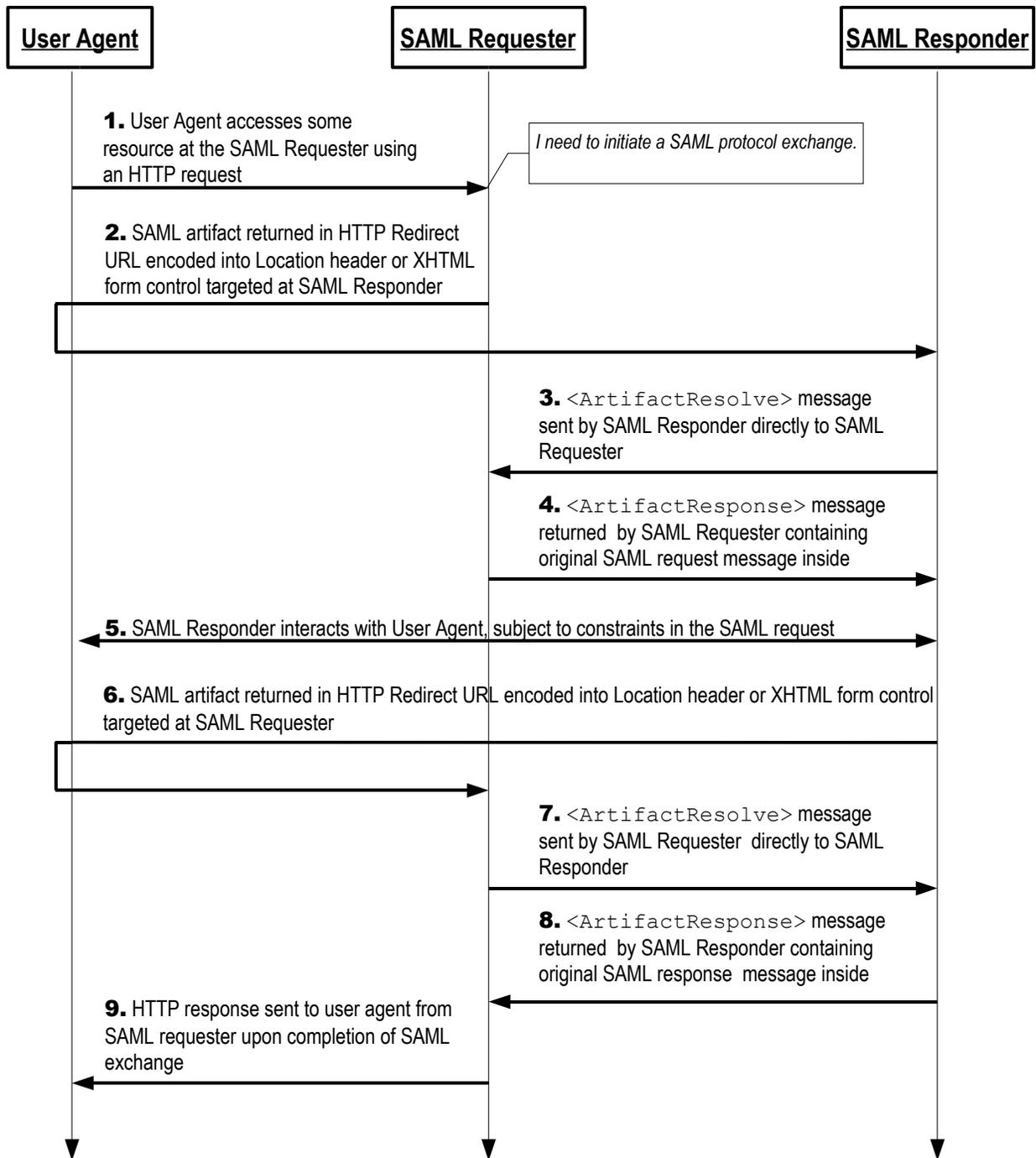
1062 Any two artifact issuers with a common receiver MUST use distinct `SourceID` values. Construction of
1063 `MessageHandle` values is governed by the principle that they SHOULD have no predictable relationship
1064 to the contents of the referenced message at the issuing site and it MUST be infeasible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

1066 **3.6.5 Message Exchange**

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

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

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



1084
1085

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

1086
1087

2. The system entity acting as a SAML requester responds to an HTTP request from the user agent by returning an artifact representing a SAML request.

- 1088 • If URL-encoded, the artifact is returned encoded into the HTTP response's Location
1089 header, and the HTTP status MUST be either 303 or 302. The SAML requester MAY
1090 include additional presentation and content in the HTTP response to facilitate the user
1091 agent's transmission of the message, as defined in HTTP 1.1 [RFC2616]. The user
1092 agent delivers the artifact by issuing an HTTP GET request to the SAML responder.
- 1093 • If form-encoded, then the artifact is returned in an XHTML document containing the
1094 form and content defined in Section 3.6.3.3. The user agent delivers the artifact by
1095 issuing an HTTP POST request to the SAML responder.
- 1096 3. The SAML responder determines the SAML requester by examining the artifact (the exact process
1097 depends on the type of artifact), and issues a `<samlp:ArtifactResolve>` request containing
1098 the artifact to the SAML requester using a direct SAML binding, temporarily reversing roles.
- 1099 4. Assuming the necessary conditions are met, the SAML requester returns a
1100 `<samlp:ArtifactResponse>` containing the original SAML request message it wishes the
1101 SAML responder to process.
- 1102 5. In general, the SAML responder MAY respond to the SAML request by immediately returning a
1103 SAML artifact or MAY return arbitrary content to facilitate subsequent interaction with the user agent
1104 necessary to fulfill the request. Specific protocols and profiles may include mechanisms to indicate
1105 the requester's level of willingness to permit this kind of interaction (for example, the `IsPassive`
1106 attribute in `<samlp:AuthnRequest>`).
- 1107 6. Eventually the responder SHOULD return a SAML artifact to the user agent to be returned to the
1108 SAML requester. The SAML response artifact is returned in the same fashion as described for the
1109 SAML request artifact in step 2. The SAML requester determines the SAML responder by examining
1110 the artifact, and issues a `<samlp:ArtifactResolve>` request containing the artifact to the SAML
1111 responder using a direct SAML binding, as in step 3.
- 1112 7. Assuming the necessary conditions are met, the SAML responder returns a
1113 `<samlp:ArtifactResponse>` containing the SAML response message it wishes the requester to
1114 process, as in step 4.
- 1115 8. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
1116 user agent.

1117 **3.6.5.1 HTTP and Caching Considerations**

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

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

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

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

1124 **3.6.5.2 Security Considerations**

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

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

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

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

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

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

1149 **3.6.6 Error Reporting**

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

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

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

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

1162 **3.6.7 Metadata Considerations**

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

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

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

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

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

```

1178     <samlp:SessionIndex>1</samlp:SessionIndex>
1179 </samlp:LogoutRequest>

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

```

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

```

1195 HTTP/1.1 302 Object Moved
1196 Date: 21 Jan 2004 07:00:49 GMT
1197 Location:
1198 https://ServiceProvider.com/SAML/SLO/Browser?SAMLart=AAQAADWNEw5VT47wcO4z
1199 X%2FiEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU%
1200 3D&RelayState=0043bfc1bc45110dae17004005b13a2b
1201 Content-Type: text/html; charset=iso-8859-1

```

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

1204 Step 3:

```

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

```

1225 Step 4:

```

1226 HTTP/1.1 200 OK
1227 Date: 21 Jan 2004 07:00:49 GMT
1228 Content-Type: text/xml
1229 Content-Length: nnnn

1230 <SOAP-ENV:Envelope
1231 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1232 <SOAP-ENV:Body>
1233 <samlp:ArtifactResponse
1234 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1235 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1236 ID="_FQvGknDfws2Z" Version="2.0"

```

```

1237         InResponseTo=" 6c3a4f8b9c2d"
1238         IssueInstant="2004-01-21T19:00:49Z">
1239         <Issuer>https://IdentityProvider.com/SAML</Issuer>
1240         <samlp:Status>
1241             <samlp:StatusCode
1242                 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1243             </samlp:Status>
1244             <samlp:LogoutRequest ID="d2b7c388cec36fa7c39c28fd298644a8"
1245                 IssueInstant="2004-01-21T19:00:49Z"
1246                 Version="2.0">
1247                 <Issuer>https://IdentityProvider.com/SAML</Issuer>
1248                 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
1249 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>
1250                 <samlp:SessionIndex>1</samlp:SessionIndex>
1251             </samlp:LogoutRequest>
1252         </samlp:ArtifactResponse>
1253     </SOAP-ENV:Body>
1254 </SOAP-ENV:Envelope>

```

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

```

1257 HTTP/1.1 302 Object Moved
1258 Date: 21 Jan 2004 07:05:49 GMT
1259 Location:
1260 https://IdentityProvider.com/SAML/SLO/Response?SAMLart=AAQAAFQIZXv5%
1261 2BQaBaE5qYurHWJ0lnAgLASqfnyidHIggbFU0mlSGFTyQiPc%
1262 3D&RelayState=0043bfc1bc45110dae17004005b13a2b
1263 Content-Type: text/html; charset=iso-8859-1

```

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

1266 Step 7:

```

1267 POST /SAML/Artifact/Resolve HTTP/1.1
1268 Host: ServiceProvider.com
1269 Content-Type: text/xml
1270 Content-Length: nnn
1271 SOAPAction: http://www.oasis-open.org/committees/security
1272 <SOAP-ENV:Envelope
1273     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1274     <SOAP-ENV:Body>
1275         <samlp:ArtifactResolve
1276             xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1277             xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1278             ID="_ec36fa7c39" Version="2.0"
1279             IssueInstant="2004-01-21T19:05:49Z">
1280             <Issuer>https://IdentityProvider.com/SAML</Issuer>
1281             <Artifact>
1282                 AAQAAFQIZXv5+QaBaE5qYurHWJ0lnAgLASqfnyidHIggbFU0mlSGFTyQiPc=
1283             </Artifact>
1284         </samlp:ArtifactResolve>
1285     </SOAP-ENV:Body>
1286 </SOAP-ENV:Envelope>

```

1287 Step 8:

```

1288 HTTP/1.1 200 OK
1289 Date: 21 Jan 2004 07:05:49 GMT
1290 Content-Type: text/xml
1291 Content-Length: nnnn
1292 <SOAP-ENV:Envelope
1293     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1294     <SOAP-ENV:Body>
1295         <samlp:ArtifactResponse
1296             xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"

```

```

1297      xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1298      ID="FQvGknDfws2Z" Version="2.0"
1299      InResponseTo="_ec36fa7c39"
1300      IssueInstant="2004-01-21T19:05:49Z">
1301      <Issuer>https://ServiceProvider.com/SAML</Issuer>
1302      <samlp:Status>
1303          <samlp:StatusCode
1304              Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1305      </samlp:Status>
1306      <samlp:LogoutResponse ID="_b0730d21b628110d8b7e004005b13a2b"
1307          InResponseTo="_d2b7c388cec36fa7c39c28fd298644a8"
1308          IssueInstant="2004-01-21T19:05:49Z"
1309          Version="2.0">
1310      <Issuer>https://ServiceProvider.com/SAML</Issuer>
1311      <samlp:Status>
1312          <samlp:StatusCode
1313              Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1314      </samlp:Status>
1315      </samlp:LogoutResponse>
1316      </samlp:ArtifactResponse>
1317      </SOAP-ENV:Body>
1318      </SOAP-ENV:Envelope>

```

1319 3.7 SAML URI Binding

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

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

1327 3.7.1 Required Information

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

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

1330 **Description:** Given below.

1331 **Updates:** None

1332 3.7.2 Protocol-Independent Aspects of the SAML URI Binding

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

1335 3.7.2.1 Basic Operation

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

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

1345 **3.7.3 Security Considerations**

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

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

1355 **3.7.4 MIME Encapsulation**

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

1359 **3.7.5 Use of HTTP URIs**

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

1363 **3.7.5.1 URI Syntax**

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

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

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

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

1373 **3.7.5.2 HTTP and Caching Considerations**

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

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

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

1379 **3.7.5.3 Security Considerations**

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

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

1384 **3.7.5.4 Error Reporting**

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

1390 **3.7.5.5 Metadata Considerations**

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

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

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

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

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

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

4 References

1406

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

1451		(SAML) V2.0. OASIS SSTC, December 2004. Document ID sstc-saml-metadata-2.0-cd-03. See http://www.oasis-open.org/committees/security/ .
1452		
1453	[SAMLmime]	application/saml+xml Media Type Registration, IETF Internet-Draft, http://www.ietf.org/internet-drafts/draft-hodges-saml-mediatype-01.txt .
1454		
1455	[SAMLSecure]	F. Hirsch et al., <i>Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0</i> . OASIS SSTC, December 2004. Document ID sstc-saml-sec-consider-2.0-cd-03. See http://www.oasis-open.org/committees/security/ .
1456		
1457		
1458		
1459	[SAMLReqs]	Darren Platt et al., SAML Requirements and Use Cases, OASIS, April 2002, http://www.oasis-open.org/committees/security/ .
1460		
1461	[SAMLWeb]	OASIS Security Services Technical Committee website, http://www.oasis-open.org/committees/security/ .
1462		
1463	[SESSION]	RL "Bob" Morgan, Support of target web server sessions in Shibboleth, http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-session-00.txt
1464		
1465		
1466	[ShibMarlena]	Marlena Erdos et al., Shibboleth Architecture DRAFT v1.1, http://shibboleth.internet2.edu/draft-internet2-shibboleth-arch-v05.html .
1467		
1468	[SOAP1.1]	D. Box et al., <i>Simple Object Access Protocol (SOAP) 1.1</i> , World Wide Web Consortium Note, May 2000, http://www.w3.org/TR/SOAP .
1469		
1470	[SOAP-PRIMER]	N. Mitra, SOAP Version 1.2 Part 0: Primer, W3C Recommendation 24 June 2003, http://www.w3.org/TR/soap12-part0/
1471		
1472	[SSL3]	A. Frier et al., <i>The SSL 3.0 Protocol</i> , Netscape Communications Corp, November 1996.
1473		
1474	[WEBSSO]	RL "Bob" Morgan, Interactions between Shibboleth and local-site web sign-on services, http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-websso-00.txt
1475		
1476		
1477	[WSS-SAML]	P. Hallam-Baker et al., <i>Web Services Security: SAML Token Profile</i> , OASIS, March 2003, http://www.oasis-open.org/committees/wss .
1478		
1479	[WSS-Sec]	A. Nadalin et al., <i>Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)</i> , OASIS Standard 200401, March 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
1480		
1481		
1482	[XMLSig]	D. Eastlake et al., <i>XML-Signature Syntax and Processing</i> , World Wide Web Consortium, http://www.w3.org/TR/xmlsig-core/ .
1483		

1484 **Appendix A. Registration of MIME media type**
1485 **application/samlassertion+xml**

1486 To: ietf-types@iana.org
1487 Subject: Registration of MIME media type `application/samlassertion+xml`

1488 **Introduction**

1489 This document defines a MIME media type -- `application/samlassertion+xml` -- for use
1490 with the XML serialization of SAML (Security Assertion Markup Language) assertions.

1491 The SAML specification sets -- [SAMLv1.0], [SAMLv1.1], [SAMLv2.0] -- are work products of the
1492 OASIS Security Services Technical Committee [SSTC]. The SAML specifications define XML-
1493 based constructs with which one may make, and convey, security assertions. Using SAML, one
1494 can assert that an authentication event pertaining to some subject has occurred and convey said
1495 assertion to a relying party, for example.

1496 SAML assertions, which are explicitly versioned, are defined by [SAMLv1Core], [SAMLv11Core],
1497 and [SAMLv2Core].

1498 **MIME media type name**

1499 `application`

1500 **MIME subtype name**

1501 `samlassertion+xml`

1502 **Required parameters**

1503 None

1504 **Optional parameters**

1505 `charset`

1506 Same as `charset` parameter of `application/xml` [RFC3023].

1507 **Encoding considerations**

1508 Same as for `application/xml` [RFC3023].

1509 **Security considerations**

1510 Per their specification, `samlassertion+xml`-typed objects do not contain executable content.
1511 However, SAML assertions are XML-based objects [XML]. As such, they have all of the general
1512 security considerations presented in section 10 of [RFC3023], as well as additional ones, since
1513 they are explicit security objects. For example, `samlassertion+xml`-typed objects will often
1514 contain data that may identify or pertain to a natural person, and may be used as a basis for
1515 sessions and access control decisions.

1516 To counter potential issues, `samlassertion+xml`-typed objects contain data that should be
1517 signed appropriately by the sender. Any such signature must be verified by the recipient of the
1518 data - both as a valid signature, and as being the signature of the sender. Issuers of
1519 `samlassertion+xml`-typed objects containing SAMLv2 assertions may also encrypt all, or

1520 portions of, the assertions (see [SAMLv2Core]).

1521 In addition, SAML profiles and protocol bindings specify use of secure channels as appropriate.

1522 [SAMLv2.0] incorporates various privacy-protection techniques in its design. For example: opaque
1523 handles, specific to interactions between specific system entities, may be assigned to subjects.
1524 The handles are mappable to wider-context identifiers (e.g. email addresses, account identifiers,
1525 etc) by only the specific parties.

1526 For a more detailed discussion of SAML security considerations and specific security-related
1527 design techniques, please refer to the SAML specifications listed in the below bibliography. The
1528 specifications containing security-specific information have been explicitly listed for each version
1529 of SAML.

1530 **Interoperability considerations**

1531 SAML assertions are explicitly versioned. Relying parties should ensure that they observe
1532 assertion version information and behave accordingly. See chapters on SAML Versioning in
1533 [SAMLv1Core], [SAMLv11Core], or [SAMLv2Core], as appropriate.

1534 **Published specification**

1535 [SAMLv2Bind] explicitly specifies use of the `application/samlassertion+xml` MIME media
1536 type. However, it is conceivable that non-SAMLv2 assertions (i.e., SAMLv1 and/or SAMLv1.1)
1537 might in practice be conveyed using SAMLv2 bindings.

1538 **Applications which use this media type**

1539 Potentially any application implementing SAML, as well as those applications implementing
1540 specifications based on SAML, e.g. those available from the Liberty Alliance [LAP].

1541 **Additional information**

1542 **Magic number(s)**

1543 In general, the same as for `application/xml` [RFC3023]. In particular, the XML root element of the
1544 returned object will have a namespace-qualified name with:

- 1545 – a local name of: `Assertion`
- 1546 – a namespace URI of: one of the version-specific SAML assertion XML
1547 namespace URIs, as defined by the appropriate version-specific SAML "core"
1548 specification (see bibliography).

1549 With SAMLv2.0 specifically, the root element of the returned object may be either
1550 `<saml:Assertion>` or `<saml:EncryptedAssertion>`, where "saml" represents any XML
1551 namespace prefix that maps to the SAMLv2.0 assertion namespace URI:

1552 `urn:oasis:names:tc:SAML:2.0:assertion`

1553 **File extension(s)**

1554 None

1555 **Macintosh File Type Code(s)**

1556 None

1557 **Person & email address to contact for further information**

1558 This registration is made on behalf of the OASIS Security Services Technical Committee (SSTC)
1559 Please refer to the SSTC website for current information on committee chairperson(s) and their
1560 contact addresses: <http://www.oasis-open.org/committees/security/>. Committee members should
1561 submit comments and potential errata to the security-services@lists.oasis-open.org list. Others
1562 should submit them by filling out the web form located at [http://www.oasis-
1563 open.org/committees/comments/form.php?wg_abbrev=security](http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=security).

1564 Additionally, the SAML developer community email distribution list, [saml-dev@lists.oasis-
1565 open.org](mailto:saml-dev@lists.oasis-open.org), may be employed to discuss usage of the application/samlassertion+xml MIME media
1566 type. The "saml-dev" mailing list is publicly archived here: [http://lists.oasis-
1567 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
1568 subscribe, send a message with the single word "subscribe" in the message body, to: [saml-dev-
1569 request@lists.oasis-open.org](mailto:saml-dev-request@lists.oasis-open.org).

1570 **Intended usage**

1571 COMMON

1572 **Author/Change controller**

1573 The SAML specification sets are a work product of the OASIS Security Services Technical
1574 Committee (SSTC). OASIS and the SSTC have change control over the SAML specification sets.

1575 **Bibliography**

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

1604		oasis-sstc-saml-bindings-1.1. http://www.oasis-open.org/committees/security/
1605		
1606	[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/
1607		
1608		
1609	[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/
1610		
1611		
1612		
1613	[SAMLv2.0]	OASIS Security Services Technical Committee, "Security Assertion Markup Language (SAML) Version 2.0 Specification Set". WORK IN PROGRESS. Available at http://www.oasis-open.org/committees/security/
1614		
1615		
1616		
1617	[SAMLv2Bind]	S. Cantor et al., "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS SSTC, August 2004. Document ID sstc-saml-bindings-2.0-cd-03, WORK IN PROGRESS. See http://www.oasis-open.org/committees/security/
1618		
1619		
1620		
1621	[SAMLv2Core]	S. Cantor et al., "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS SSTC, August 2004. Document ID sstc-saml-core-2.0-cd-03, WORK IN PROGRESS. See http://www.oasis-open.org/committees/security/
1622		
1623		
1624		
1625	[SAMLv2Prof]	S. Cantor et al., "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS SSTC, August 2004. Document ID sstc-saml-profiles-2.0-cd-03, WORK IN PROGRESS. See http://www.oasis-open.org/committees/security/
1626		
1627		
1628		
1629	[SAMLv2Sec]	F. Hirsch et al., "Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS SSTC, August 2004, WORK IN PROGRESS. Document ID sstc-saml-sec-consider-2.0-cd-03. See http://www.oasis-open.org/committees/security/
1630		
1631		
1632		
1633	[SSTC]	"OASIS Security Services Technical Committee". See http://www.oasis-open.org/committees/security/
1634		
1635	[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/
1636		
1637		
1638		

1639 Appendix B. Acknowledgments

1640 The editors would like to acknowledge the contributions of the OASIS Security Services Technical
1641 Committee, whose voting members at the time of publication were:

- 1642 • Conor Cahill, AOL
- 1643 • John Hughes, Atos Origin
- 1644 • Hal Lockhart, BEA Systems
- 1645 • Mike Beach, Boeing
- 1646 • Rebekah Metz, Booz Allen Hamilton
- 1647 • Rick Randall, Booz Allen Hamilton
- 1648 • Ronald Jacobson, Computer Associates
- 1649 • Paul Madsen, Entrust
- 1650 • Dana Kaufman, Forum Systems
- 1651 • Paula Austel, IBM
- 1652 • Michael McIntosh, IBM
- 1653 • Anthony Nadalin, IBM
- 1654 • Nick Ragouzis, Individual
- 1655 • Scott Cantor, Internet2
- 1656 • Bob Morgan, Internet2
- 1657 • Peter Davis, Neustar
- 1658 • Jeff Hodges, Neustar
- 1659 • Frederick Hirsch, Nokia
- 1660 • John Kemp, Nokia
- 1661 • Abbie Barbir, Nortel Networks
- 1662 • Scott Kiestler, Novell
- 1663 • Cameron Morris, Novell
- 1664 • Charles Knouse, Oblix
- 1665 • Steve Anderson, OpenNetwork
- 1666 • Ari Kermaier, Oracle
- 1667 • Vamsi Motukuru, Oracle
- 1668 • Darren Platt, Ping Identity
- 1669 • Prateek Mishra, Principal Identity
- 1670 • Jim Lien, RSA Security
- 1671 • Rob Philpott, RSA Security
- 1672 • Dipak Chopra, SAP
- 1673 • Jahan Moreh, Sigaba
- 1674 • Bhavna Bhatnagar, Sun Microsystems
- 1675 • Eve Maler, Sun Microsystems
- 1676 • Ronald Monzillo, Sun Microsystems
- 1677 • Emily Xu, Sun Microsystems
- 1678 • Greg Whitehead, Trustgenix

1679 The editors also would like to acknowledge the following people for their contributions to previous versions
1680 of the OASIS Security Assertions Markup Language Standard:

- 1681 • Stephen Farrell, Baltimore Technologies
- 1682 • David Orchard, BEA Systems
- 1683 • Krishna Sankar, Cisco Systems
- 1684 • Zahid Ahmed, CommerceOne
- 1685 • Carlisle Adams, Entrust
- 1686 • Tim Moses, Entrust
- 1687 • Nigel Edwards, Hewlett-Packard
- 1688 • Joe Pato, Hewlett-Packard
- 1689 • Bob Blakley, IBM
- 1690 • Marlena Erdos, IBM
- 1691 • Marc Chanliau, Netegrity
- 1692 • Chris McLaren, Netegrity
- 1693 • Lynne Rosenthal, NIST
- 1694 • Mark Skall, NIST
- 1695 • Simon Godik, Overxeer
- 1696 • Charles Norwood, SAIC
- 1697 • Evan Prodromou, Securant
- 1698 • Robert Griffin, RSA Security (former editor)
- 1699 • Sai Allarvarpu, Sun Microsystems
- 1700 • Chris Ferris, Sun Microsystems
- 1701 • Emily Xu, Sun Microsystems
- 1702 • Mike Myers, Traceroute Security
- 1703 • Phillip Hallam-Baker, VeriSign (former editor)
- 1704 • James Vanderbeek, Vodafone
- 1705 • Mark O'Neill, Vordel
- 1706 • Tony Palmer, Vordel

1707 Finally, the editors wish to acknowledge the following people for their contributions of material used as
1708 input to the OASIS Security Assertions Markup Language specifications:

- 1709 • Thomas Gross, IBM
- 1710 • Birgit Pfitzmann, IBM

1711 Appendix C. Notices

1712 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that
1713 might be claimed to pertain to the implementation or use of the technology described in this document or
1714 the extent to which any license under such rights might or might not be available; neither does it represent
1715 that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to
1716 rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made
1717 available for publication and any assurances of licenses to be made available, or the result of an attempt
1718 made to obtain a general license or permission for the use of such proprietary rights by implementors or
1719 users of this specification, can be obtained from the OASIS Executive Director.

1720 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or
1721 other proprietary rights which may cover technology that may be required to implement this specification.
1722 Please address the information to the OASIS Executive Director.

1723 **Copyright © OASIS Open 2004. All Rights Reserved.**

1724 This document and translations of it may be copied and furnished to others, and derivative works that
1725 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and
1726 distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and
1727 this paragraph are included on all such copies and derivative works. However, this document itself may
1728 not be modified in any way, such as by removing the copyright notice or references to OASIS, except as
1729 needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights
1730 defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it
1731 into languages other than English.

1732 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors
1733 or assigns.

1734 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1735 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
1736 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
1737 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.