



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

Committee Draft 04, 15 January 2005

Document identifier:

sstc-saml-bindings-2.0-cd-04

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
Thomas Wisniewski, 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
Paul Madsen, NTT
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

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

49 **Abstract:**

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

52 **Status:**

53 This is a **Committee Draft** approved by the Security Services Technical Committee on 15
54 January 2005.

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

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

64 Table of Contents

65	1 Introduction.....	5
66	1.1 Protocol Binding Concepts.....	5
67	1.2 Notation.....	5
68	2 Guidelines for Specifying Additional Protocol Bindings.....	7
69	3 Protocol Bindings.....	8
70	3.1 General Considerations.....	8
71	3.1.1 Use of RelayState.....	8
72	3.1.2 Security.....	8
73	3.1.2.1 Use of SSL 3.0 or TLS 1.0.....	8
74	3.1.2.2 Data Origin Authentication.....	8
75	3.1.2.3 Message Integrity.....	8
76	3.1.2.4 Message Confidentiality.....	9
77	3.1.2.5 Security Considerations.....	9
78	3.2 SAML SOAP Binding.....	9
79	3.2.1 Required Information.....	9
80	3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding.....	10
81	3.2.2.1 Basic Operation.....	10
82	3.2.2.2 SOAP Headers.....	10
83	3.2.3 Use of SOAP over HTTP.....	11
84	3.2.3.1 HTTP Headers.....	11
85	3.2.3.2 Caching.....	11
86	3.2.3.3 Error Reporting.....	11
87	3.2.3.4 Metadata Considerations.....	12
88	3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP.....	12
89	3.3 Reverse SOAP (PAOS) Binding.....	13
90	3.3.1 Required Information.....	13
91	3.3.2 Overview.....	13
92	3.3.3 Message Exchange.....	13
93	3.3.3.1 HTTP Request, SAML Request in SOAP Response.....	14
94	3.3.3.2 SAML Response in SOAP Request, HTTP Response.....	15
95	3.3.4 Caching.....	15
96	3.3.5 Security Considerations.....	15
97	3.3.5.1 Error Reporting.....	15
98	3.3.5.2 Metadata Considerations.....	15
99	3.4 HTTP Redirect Binding.....	15
100	3.4.1 Required Information.....	16
101	3.4.2 Overview.....	16
102	3.4.3 RelayState.....	16
103	3.4.4 Message Encoding.....	16
104	3.4.4.1 DEFLATE Encoding.....	17
105	3.4.5 Message Exchange.....	18
106	3.4.5.1 HTTP and Caching Considerations.....	19
107	3.4.5.2 Security Considerations.....	19
108	3.4.6 Error Reporting.....	20
109	3.4.7 Metadata Considerations.....	20
110	3.4.8 Example SAML Message Exchange Using HTTP Redirect.....	20

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

1 Introduction

155

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

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

1.1 Protocol Binding Concepts

163

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

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

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

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

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

1.2 Notation

178

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

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

183 `Example code listings appear like this.`

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

185 Conventional XML namespace prefixes are used throughout this specification to stand for their respective
186 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 [SOAP11].

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

2 Guidelines for Specifying Additional Protocol Bindings

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

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

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

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

224 **3 Protocol Bindings**

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

226 **3.1 General Considerations**

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

228 **3.1.1 Use of RelayState**

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

236 **3.1.2 Security**

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

239 **3.1.2.1 Use of SSL 3.0 or TLS 1.0**

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

244 **3.1.2.2 Data Origin Authentication**

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

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

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

254 **3.1.2.3 Message Integrity**

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

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

260 3.1.2.4 Message Confidentiality

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

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

266 3.1.2.5 Security Considerations

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

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

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

274 3.2 SAML SOAP Binding

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

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

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

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

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

293 3.2.1 Required Information

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

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

296 **Description:** Given below.

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

298 3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding

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

302 3.2.2.1 Basic Operation

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

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

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

- 309 1. A system entity acting as a SAML requester transmits a SAML request element within the body of
310 a SOAP message to a system entity acting as a SAML responder. The SAML requester MUST
311 NOT include more than one SAML request per SOAP message or include any additional XML
312 elements in the SOAP body.
- 313 2. The SAML responder MUST return either a SAML response element within the body of another
314 SOAP message or generate a SOAP fault. The SAML responder MUST NOT include more than
315 one SAML response per SOAP message or include any additional XML elements in the SOAP
316 body. If a SAML responder cannot, for some reason, process a SAML request, it MUST generate a
317 SOAP fault. SOAP fault codes MUST NOT be sent for errors within the SAML problem domain, for
318 example, inability to find an extension schema or as a signal that the subject is not authorized to
319 access a resource in an authorization query. (SOAP 1.1 faults and fault codes are discussed in
320 [SOAP11] §4.1.)

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

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

329 3.2.2.2 SOAP Headers

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

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

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

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

341 3.2.3 Use of SOAP over HTTP

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

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

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

349 3.2.3.1 HTTP Headers

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

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

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

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

361 3.2.3.2 Caching

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

364 When using HTTP 1.1, requesters SHOULD:

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

367 When using HTTP 1.1, responders SHOULD:

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

372 3.2.3.3 Error Reporting

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

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

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

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

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

387 3.2.3.4 Metadata Considerations

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

391 3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP

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

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

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

```
414 HTTP/1.1 200 OK
415 Content-Type: text/xml
416 Content-Length: nnnn
417 <SOAP-ENV:Envelope
418   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
419   <SOAP-ENV:Body>
420     <samlp:Response xmlns:samlp="..." xmlns:saml="..." xmlns:ds="..."
421   ID="_6c3a4f8b9c2d" Version="2.0" IssueInstant="2004-03-27T08:42:00Z">
422     <saml:Issuer>https://www.example.com/SAML</saml:Issuer>
423     <ds:Signature> ... </ds:Signature>
424     <Status>
425       <StatusCode Value="..." />
426     </Status>
427
428     <saml:Assertion>
429       <saml:Subject>
430       ...
431       </saml:Subject>
432       <saml:AttributeStatement>
433       ...
434       </saml:AttributeStatement>
435     </saml:Assertion>
```

```
436     </samlp:Response>
437     </SOAP-Env:Body>
438 </SOAP-ENV:Envelope>
```

439 **3.3 Reverse SOAP (PAOS) Binding**

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

443 **3.3.1 Required Information**

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

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

446 **Description:** Given below.

447 **Updates:** None.

448 **3.3.2 Overview**

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

456 **3.3.3 Message Exchange**

457 The PAOS binding includes two component message exchange patterns:

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

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

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

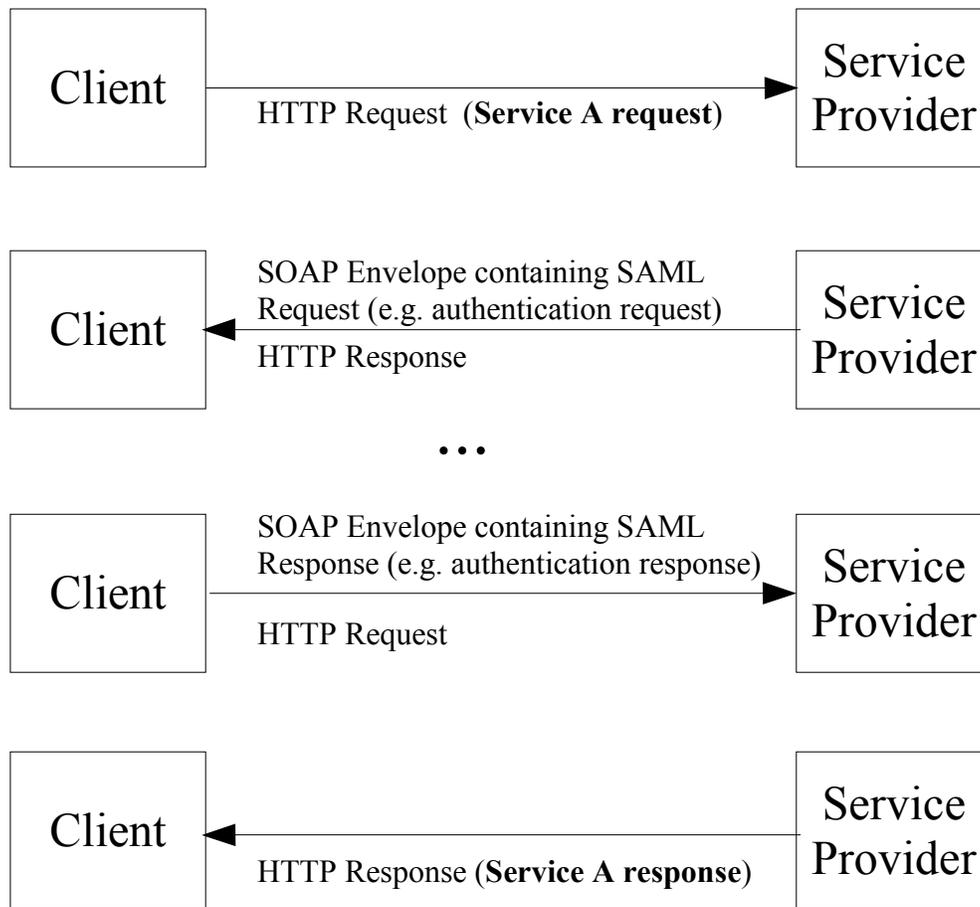


Figure 1: PAOS Binding Message Exchanges

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

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

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

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

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

484 3.3.3.1 HTTP Request, SAML Request in SOAP Response

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

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

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

492 **3.3.3.2 SAML Response in SOAP Request, HTTP Response**

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

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

500 **3.3.4 Caching**

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

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

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

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

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

511 **3.3.5 Security Considerations**

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

515 **3.3.5.1 Error Reporting**

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

519 **3.3.5.2 Metadata Considerations**

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

523 **3.4 HTTP Redirect Binding**

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

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

533 3.4.1 Required Information

534 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect
535 **Contact information:** security-services-comment@lists.oasis-open.org
536 **Description:** Given below.
537 **Updates:** None.

538 3.4.2 Overview

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

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

547 3.4.3 RelayState

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

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

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

561 3.4.4 Message Encoding

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

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

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

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

576 3.4.4.1 DEFLATE Encoding

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

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

- 581 1. Any signature on the SAML protocol message, including the `<ds:Signature>` XML element itself,
582 MUST be removed. Note that if the content of the message includes another signature, such as a
583 signed SAML assertion, this embedded signature is not removed. However, the length of such a
584 message after encoding essentially precludes using this mechanism. Thus SAML protocol
585 messages that contain signed content SHOULD NOT be encoded using this mechanism.
- 586 2. The DEFLATE compression mechanism, as specified in [RFC1951] is then applied to the entire
587 remaining XML content of the original SAML protocol message.
- 588 3. The compressed data is subsequently base64-encoded according to the rules specified in IETF
589 RFC 2045 [RFC2045]. Linefeeds or other whitespace MUST be removed from the result.
- 590 4. The base-64 encoded data is then URL-encoded, and added to the URL as a query string
591 parameter which MUST be named `SAMLRequest` (if the message is a SAML request) or
592 `SAMLResponse` (if the message is a SAML response).
- 593 5. If RelayState data is to accompany the SAML protocol message, it MUST be URL-encoded and
594 placed in an additional query string parameter named `RelayState`.
- 595 6. If the original SAML protocol message was signed using an XML digital signature, a new signature
596 covering the encoded data as specified above MUST be attached using the rules stated below.

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

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

```
607 SAMLRequest=value&RelayState=value&SigAlg=value  
608 SAMLResponse=value&RelayState=value&SigAlg=value
```

- 609 3. The resulting string of bytes is the octet string to be fed into the signature algorithm. Any other
610 content in the original query string is not included and not signed.
- 611 4. The signature value MUST be encoded using the base64 encoding (see RFC 2045 [RFC2045]) with
612 any whitespace removed, and included as a query string parameter named `Signature`. Note that
613 some characters in the base64-encoded signature value may themselves require URL-encoding
614 before being added.

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

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

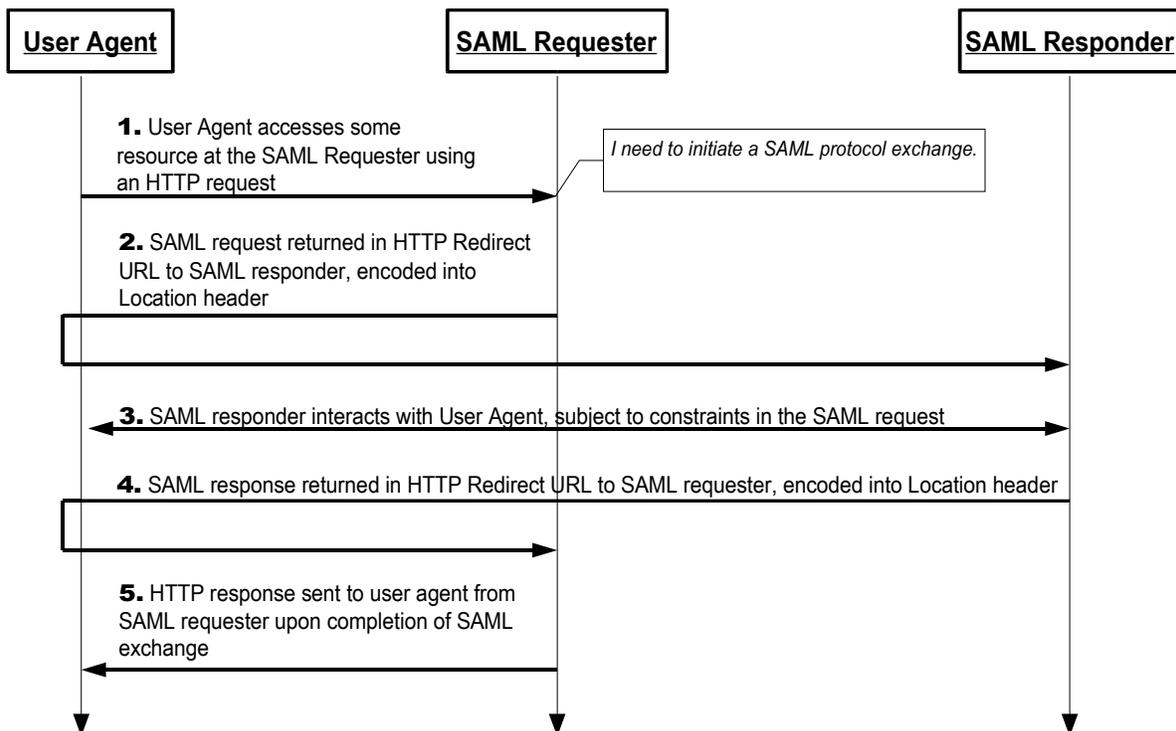
619 Note that when verifying signatures, the order of the query string parameters on the resulting URL to be
620 verified is not prescribed by this binding. The parameters may appear in any order. Before verifying a
621 signature, if any, the relying party MUST insure that the parameter values to be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

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

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

629 3.4.5 Message Exchange

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



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

637 2. The system entity acting as a SAML requester responds to the HTTP request from the user agent in
638 step 1 by returning a SAML request. The SAML request is returned encoded into the HTTP

639 response's Location header, and the HTTP status MUST be either 303 or 302. The SAML requester
640 MAY include additional presentation and content in the HTTP response to facilitate the user agent's
641 transmission of the message, as defined in HTTP 1.1 [RFC2616]. The user agent delivers the
642 SAML request by issuing an HTTP GET request to the SAML responder.

- 643 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
644 SAML response or MAY return arbitrary content to facilitate subsequent interaction with the user
645 agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to
646 indicate the requester's level of willingness to permit this kind of interaction (for example, the
647 `IsPassive` attribute in `<samlp:AuthnRequest>`).
- 648 4. Eventually the responder SHOULD return a SAML response to the user agent to be returned to the
649 SAML requester. The SAML response is returned in the same fashion as described for the SAML
650 request in step 2.
- 651 5. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
652 user agent.

653 **3.4.5.1 HTTP and Caching Considerations**

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

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

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

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

660 **3.4.5.2 Security Considerations**

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

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

668 This binding SHOULD NOT be used if the content of the request or response should not be exposed to
669 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is
670 OPTIONAL and depends on the environment of use. If confidentiality is necessary, SSL 3.0 [SSL3] or TLS
671 1.0 [RFC2246] SHOULD be used to protect the message in transit between the user agent and the SAML
672 requester and responder.

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

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

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

681 3.4.6 Error Reporting

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

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

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

689 3.4.7 Metadata Considerations

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

693 3.4.8 Example SAML Message Exchange Using HTTP Redirect

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

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

```
697 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
698 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
699 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
700 21T19:00:49Z" Version="2.0">  
701 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
702 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
703 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
704 <samlp:SessionIndex>1</samlp:SessionIndex>  
705 </samlp:LogoutRequest>
```

```
706 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
707 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
708 ID="b0730d21b628110d8b7e004005b13a2b"  
709 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"  
710 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">  
711 <Issuer>https://ServiceProvider.com/SAML</Issuer>  
712 <samlp:Status>  
713 <samlp:StatusCode  
714 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>  
715 </samlp:Status>  
716 </samlp:LogoutResponse>
```

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

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

```
725 Location:
726 https://ServiceProvider.com/SAML/SLO/Browser?SAMLRequest=fVFdS8MwFH0f7D%
727 2BUvGdNsq62oSsIQyhMESc%2B%2BJYlRbWpObeyvz3puv2IMjyFM7HPedyK1DdsZdb%2F%
728 2BEHfLFfgwVMtT3RgTwzazIEJ72CFqRTnQWJWu7uH7dSLJjsg0ev%2FZFMlttiBWADtt6R%
729 2BSyJr9msiRH7070sCm31Mj%2Bo%2BC%
730 2B1KA5GLEWeZaogSQMw2MYBKodrIhjLKONU8FdeSsZkVr6T5M0GiHMjvWCknqZXZ2OoPx7kG
731 naGOuwXZ%2Fn4L9bY8NC%
732 2By4dulXpRXnxPcXizSZ58KfTeHujEWkNPZylsh9bAMYUjO2Uiy3jCpTCMo5M1StVjmN9SO1
733 50s191U6RV2Dp0vsLIy7NM7YU82r9B90PrvCf85W%2FwL8zSVQzAEAAA%3D%
734 3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F%
735 2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-
736 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE
737 Content-Type: text/html; charset=iso-8859-1
```

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

```
742 HTTP/1.1 302 Object Moved
743 Date: 21 Jan 2004 07:00:49 GMT
744 Location:
745 https://IdentityProvider.com/SAML/SLO/Response?SAMLResponse=fVFNa4QwEL0X%
746 2Bh8k912TaDUGFUp7EbZQ6rKH3mKcbQVNJBOX%2FvxaXQ9tYec0vHlv3nzKqIZ%2B1Af7YSf%
747 2FBjhagxB8Db1BuZQKMjkjrcIOpVEDoPRa1o8vB8n3VI7Oeqtt1bJbbJCB0c7a8j9XTBH9Vy
748 QhqYRbTlrEi4Yo61oUqA0pvShYZHiDQkqs411tAVpeZPqSagNOkrOas4zzcW55Zl14liJrTXi
749 BJVBr4wvCJ877ijbcXZkmaRUxtk7CU7gcB5mLu8pKVddvghd%
750 2Ben9iDIMA3CXtsOrs5euBbfxdgh%2F9snDK%2FEqW69Ye%2BUnvGL%2F8CfbQnBS%
751 2FQS3z4QLW9aT1oBIws0j%2FGoyAb9%2FV34Dw5k779IBAAA%
752 3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F%
753 2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-
754 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE
755 Content-Type: text/html; charset=iso-8859-1
```

756 3.5 HTTP POST Binding

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

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

762 3.5.1 Required Information

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

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

765 **Description:** Given below.

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

768 3.5.2 Overview

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

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

777 **3.5.3 RelayState**

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

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

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

791 **3.5.4 Message Encoding**

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

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

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

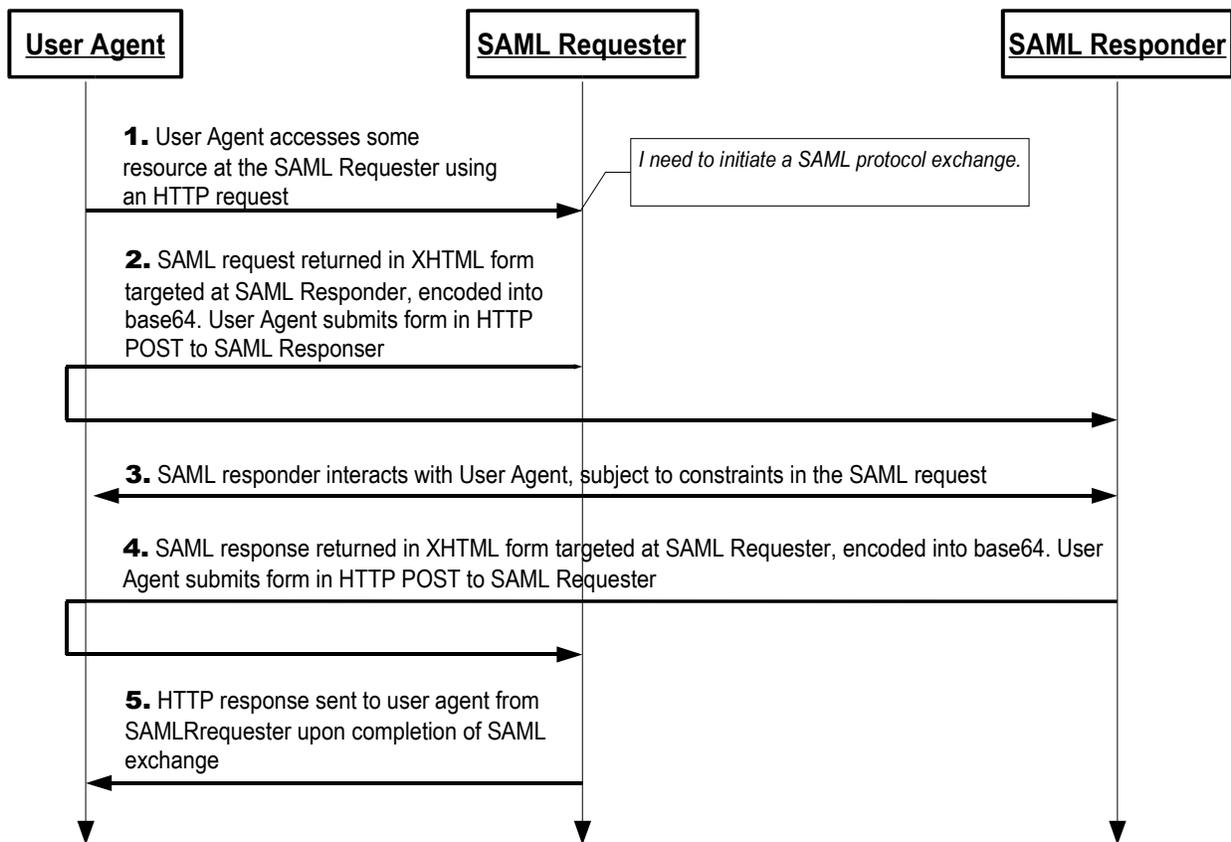
804 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using
805 this binding to which the SAML message is to be delivered. The `method` attribute MUST be "POST".

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

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

812 **3.5.5 Message Exchange**

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



- 818 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
819 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 820 2. The system entity acting as a SAML requester responds to an HTTP request from the user agent by
821 returning a SAML request. The request is returned in an XHTML document containing the form and
822 content defined in Section 3.5.4. The user agent delivers the SAML request by issuing an HTTP
823 POST request to the SAML responder.
- 824 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
825 SAML response or MAY return arbitrary content to facilitate subsequent interaction with the user
826 agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to
827 indicate the requester's level of willingness to permit this kind of interaction (for example, the
828 `IsPassive` attribute in `<samlp:AuthnRequest>`).
- 829 4. Eventually the responder SHOULD return a SAML response to the user agent to be returned to the
830 SAML requester. The SAML response is returned in the same fashion as described for the SAML
831 request in step 2.
- 832 5. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
833 user agent.

834 3.5.5.1 HTTP and Caching Considerations

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

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

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

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

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

841 **3.5.5.2 Security Considerations**

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

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

850 This binding SHOULD NOT be used if the content of the request or response should not be exposed to
851 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is
852 OPTIONAL and depends on the environment of use. If confidentiality is necessary, SSL 3.0 [SSL3] or TLS
853 1.0 [RFC2246] SHOULD be used to protect the message in transit between the user agent and the SAML
854 requester and responder.

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

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

864 **3.5.6 Error Reporting**

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

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

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

872 **3.5.7 Metadata Considerations**

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

876 **3.5.8 Example SAML Message Exchange Using HTTP POST**

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

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

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

995 **Description:** Given below.

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

998 **3.6.2 Overview**

999 The HTTP Artifact binding is intended for cases in which the SAML requester and responder need to
1000 communicate using an HTTP user agent as an intermediary, but the intermediary's limitations preclude or
1001 discourage the transmission of an entire message (or message exchange) through it. This may be for
1002 technical reasons or because of a reluctance to expose the message content to the intermediary (and if
1003 the use of encryption is not practical).

1004 Note that because of the need to subsequently resolve the artifact using another synchronous binding,
1005 such as SOAP, a direct communication path must exist between the SAML message sender and recipient
1006 in the reverse direction of the artifact's transmission (the receiver of the message and artifact must be
1007 able to send a `<samlp:ArtifactResolve>` request back to the artifact issuer). The artifact issuer must
1008 also maintain state while the artifact is pending, which has implications for load-balanced environments.

1009 **3.6.3 Message Encoding**

1010 There are two methods of encoding an artifact for use with this binding. One is to encode the artifact into a
1011 URL parameter and the other is to place the artifact in an HTML form control. When URL encoding is
1012 used, the HTTP GET method is used to deliver the message, while POST is used with form encoding. All
1013 endpoints that support this binding MUST support both techniques.

1014 **3.6.3.1 RelayState**

1015 RelayState data MAY be included with a SAML artifact transmitted with this binding. The value MUST
1016 NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the message
1017 independent of any other protections that may or may not exist during message transmission. Signing is
1018 not realistic given the space limitation, but because the value is exposed to third-party tampering, the
1019 entity SHOULD insure that the value has not been tampered with by using a checksum, a pseudo-random
1020 value, or similar means.

1021 If an artifact that represents a SAML request is accompanied by RelayState data, then the SAML
1022 responder MUST return its SAML protocol response using a binding that also supports a RelayState
1023 mechanism, and it MUST place the exact data it received with the artifact into the corresponding
1024 RelayState parameter in the response.

1025 If no such value is included with an artifact representing a SAML request, or if the SAML response
1026 message is being generated without a corresponding request, then the SAML responder MAY include
1027 RelayState data to be interpreted by the recipient based on the use of a profile or prior agreement
1028 between the parties.

1029 **3.6.3.2 URL Encoding**

1030 To encode an artifact into a URL, the artifact value is URL-encoded and placed in a query string
1031 parameter named `SAMLart`.

1032 If a "RelayState" value is to accompany the SAML artifact, it MUST be URL-encoded and placed in an
1033 additional query string parameter named `RelayState`.

1034 3.6.3.3 Form Encoding

1035 A SAML artifact is form-encoded by placing it in a hidden form control within a form as defined by
1036 [HTML401], chapter 17. The HTML document MUST adhere to the XHTML specification, [XHTML]. The
1037 form control MUST be named `SAMLart`. Any additional form controls or presentation MAY be included but
1038 MUST NOT be required in order for the recipient to process the artifact.

1039 If a "RelayState" value is to accompany the SAML artifact, it MUST be placed in an additional hidden form
1040 control named `RelayState`, within the same form with the SAML message.

1041 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using
1042 this binding to which the artifact is to be delivered. The `method` attribute MUST be set to "POST".

1043 Any technique supported by the user agent MAY be used to cause the submission of the form, and any
1044 form content necessary to support this MAY be included, such as submit controls and client-side scripting
1045 commands. However, the recipient MUST be able to process the artifact without regard for the
1046 mechanism by which the form submission is initiated.

1047 Note that any form control values included MUST be transformed so as to be safe to include in the
1048 XHTML document. This includes transforming characters such as quotes into HTML entities, etc.

1049 3.6.4 Artifact Format

1050 With respect to this binding, an artifact is a short, opaque string. Different types can be defined and used
1051 without affecting the binding. The important characteristics are the ability of an artifact receiver to identify
1052 the issuer of the artifact, resistance to tampering and forgery, uniqueness, and compactness.

1053 The general format of any artifact includes a mandatory two-byte artifact type code and a two-byte index
1054 value identifying a specific endpoint of the artifact resolution service of the issuer, as follows:

```
1055 SAML_artifact      := B64 (TypeCode EndpointIndex RemainingArtifact)
1056 TypeCode           := Byte1Byte2
1057 EndpointIndex      := Byte1Byte2
```

1058 The notation `B64 (TypeCode EndpointIndex RemainingArtifact)` stands for the application of
1059 the base64 [RFC2045] transformation to the catenation of the `TypeCode`, `EndpointIndex`, and
1060 `RemainingArtifact`.

1061 The following practices are RECOMMENDED for the creation of SAML artifacts:

- 1062 • Each issuer is assigned an identifying URI, also known as the issuer's entity (or provider) ID. See
1063 Section 8.3.6 of [SAMLCore] for a discussion of this kind of identifier.
- 1064 • The issuer constructs the `SourceID` component of the artifact by taking the SHA-1 hash of the
1065 identification URL. The hash value is NOT encoded into hexadecimal.
- 1066 • The `MessageHandle` value is constructed from a cryptographically strong random or
1067 pseudorandom number sequence [RFC1750] generated by the issuer. The sequence consists of
1068 values of at least 16 bytes in size. These values should be padded as needed to a total length of 20
1069 bytes.

1070 The following describes the single artifact type defined by SAML V2.0.

1071 3.6.4.1 Required Information

1072 **Identification:** urn:oasis:names:tc:SAML:2.0:artifact-04

1073 **Contact information:** security-services-comment@lists.oasis-open.org

1074 **Description:** Given below.

1075 **Updates:** None.

1076 **3.6.4.2 Format Details**

1077 SAML V2.0 defines an artifact type of type code 0x0004. This artifact type is defined as follows:

1078	TypeCode	:= 0x0004
1079	RemainingArtifact	:= SourceID MessageHandle
1080	SourceID	:= 20-byte_sequence
1081	MessageHandle	:= 20-byte_sequence

1082 SourceID is a 20-byte sequence used by the artifact receiver to determine artifact issuer identity and the
1083 set of possible resolution endpoints.

1084 It is assumed that the destination site will maintain a table of SourceID values as well as one or more
1085 indexed URL endpoints (or addresses) for the corresponding SAML responder. The SAML metadata
1086 specification [SAMLMeta] MAY be used for this purpose. On receiving the SAML artifact, the receiver
1087 determines if the SourceID belongs to a known artifact issuer and obtains the location of the SAML
1088 responder using the EndpointIndex before sending a SAML <samlp:ArtifactResolve> message
1089 to it.

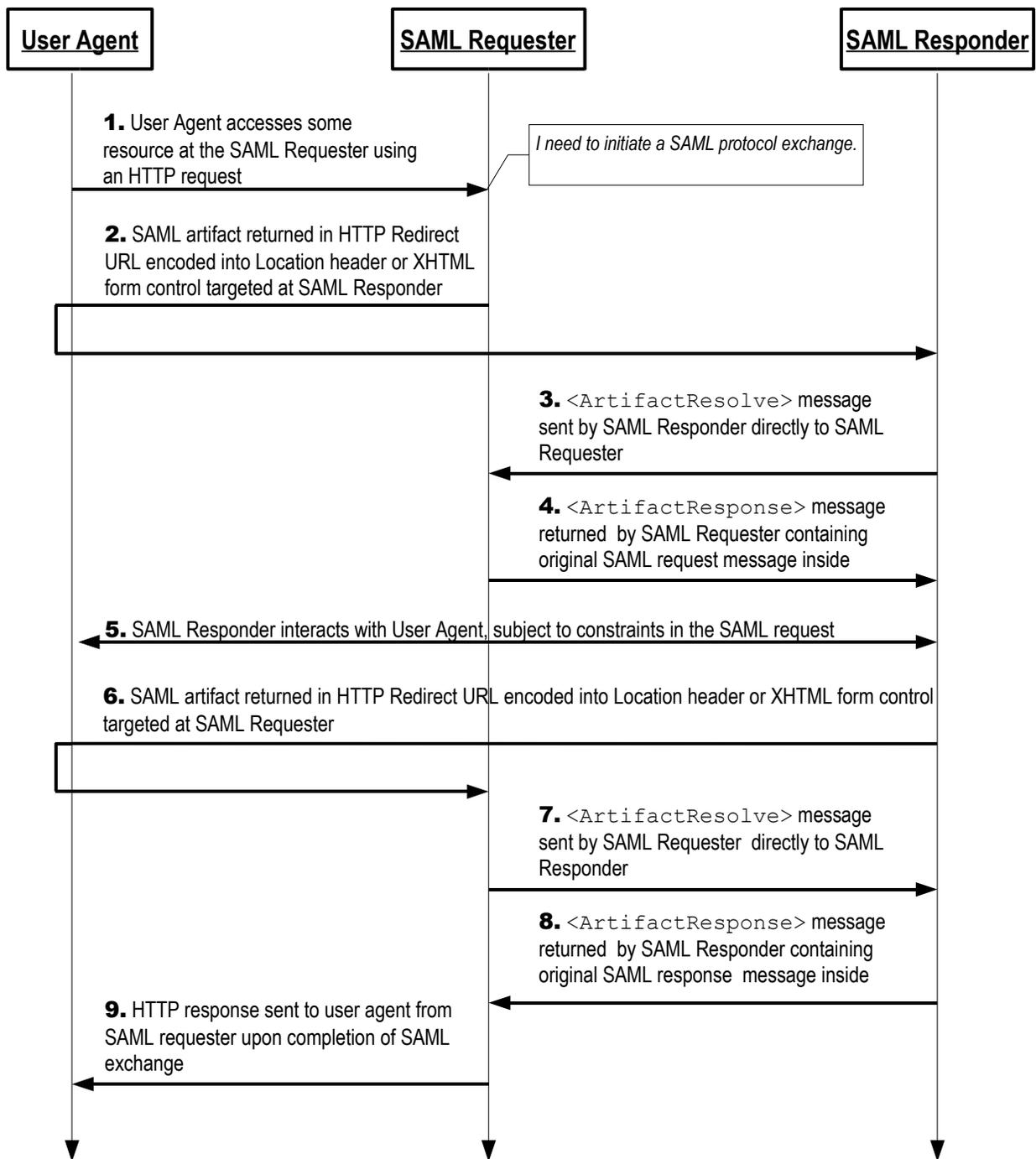
1090 Any two artifact issuers with a common receiver MUST use distinct SourceID values. Construction of
1091 MessageHandle values is governed by the principle that they SHOULD have no predictable relationship
1092 to the contents of the referenced message at the issuing site and it MUST be infeasible to construct or
1093 guess the value of a valid, outstanding message handle.

1094 **3.6.5 Message Exchange**

1095 The system model used for SAML conversations by means of this binding is a request-response model in
1096 which an artifact reference takes the place of the actual message content, and the artifact reference is
1097 sent to the user agent in an HTTP response and delivered to the message recipient in an HTTP request.
1098 The HTTP interactions before, between, and after these exchanges take place is unspecified. Both the
1099 SAML requester and responder are assumed to be HTTP responders.

1100 Additionally, it is assumed that on receipt of an artifact by way of the user agent, the recipient invokes a
1101 separate, direct exchange with the artifact issuer using the Artifact Resolution Protocol defined in
1102 [SAMLCore]. This exchange MUST use a binding that does not use the HTTP user agent as an
1103 intermediary, such as the SOAP binding. On the successful acquisition of a SAML protocol message, the
1104 artifact is discarded and the processing of the primary SAML protocol exchange resumes (or ends, if the
1105 message is a response).

1106 Issuing and delivering an artifact, along with the subsequent resolution step, constitutes half of the overall
1107 SAML protocol exchange. This binding can be used to deliver either or both halves of a SAML protocol
1108 exchange. A binding composable with it, such as the HTTP Redirect (see Section 3.4) or POST (see
1109 Section 3.5) binding, MAY be used to carry the other half of the exchange. The following sequence
1110 assumes that the artifact binding is used for both halves. See the diagram below illustrating the messages
1111 exchanged.



- 1112 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
 1113 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 1114 2. The system entity acting as a SAML requester responds to an HTTP request from the user agent by
 1115 returning an artifact representing a SAML request.
- 1116 • If URL-encoded, the artifact is returned encoded into the HTTP response's Location
 1117 header, and the HTTP status MUST be either 303 or 302. The SAML requester MAY
 1118 include additional presentation and content in the HTTP response to facilitate the user
 1119 agent's transmission of the message, as defined in HTTP 1.1 [RFC2616]. The user

- 1120 agent delivers the artifact by issuing an HTTP GET request to the SAML responder.
- 1121 • If form-encoded, then the artifact is returned in an XHTML document containing the
1122 form and content defined in Section 3.6.3.3. The user agent delivers the artifact by
1123 issuing an HTTP POST request to the SAML responder.
- 1124 3. The SAML responder determines the SAML requester by examining the artifact (the exact process
1125 depends on the type of artifact), and issues a `<samlp:ArtifactResolve>` request containing
1126 the artifact to the SAML requester using a direct SAML binding, temporarily reversing roles.
- 1127 4. Assuming the necessary conditions are met, the SAML requester returns a
1128 `<samlp:ArtifactResponse>` containing the original SAML request message it wishes the
1129 SAML responder to process.
- 1130 5. In general, the SAML responder MAY respond to the SAML request by immediately returning a
1131 SAML artifact or MAY return arbitrary content to facilitate subsequent interaction with the user agent
1132 necessary to fulfill the request. Specific protocols and profiles may include mechanisms to indicate
1133 the requester's level of willingness to permit this kind of interaction (for example, the `IsPassive`
1134 attribute in `<samlp:AuthnRequest>`).
- 1135 6. Eventually the responder SHOULD return a SAML artifact to the user agent to be returned to the
1136 SAML requester. The SAML response artifact is returned in the same fashion as described for the
1137 SAML request artifact in step 2. The SAML requester determines the SAML responder by examining
1138 the artifact, and issues a `<samlp:ArtifactResolve>` request containing the artifact to the SAML
1139 responder using a direct SAML binding, as in step 3.
- 1140 7. Assuming the necessary conditions are met, the SAML responder returns a
1141 `<samlp:ArtifactResponse>` containing the SAML response message it wishes the requester to
1142 process, as in step 4.
- 1143 8. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
1144 user agent.

1145 **3.6.5.1 HTTP and Caching Considerations**

1146 HTTP proxies and the user agent intermediary should not cache SAML artifacts. To ensure this, the
1147 following rules SHOULD be followed.

1148 When returning SAML artifacts using HTTP 1.1, HTTP responders SHOULD:

- 1149 • Include a `Cache-Control` header field set to "no-cache, no-store".
1150 • Include a `Pragma` header field set to "no-cache".

1151 There are no other restrictions on the use of HTTP headers.

1152 **3.6.5.2 Security Considerations**

1153 This binding uses a combination of indirect transmission of a message reference followed by a direct
1154 exchange to return the actual message. As a result, the message reference (artifact) need not itself be
1155 authenticated or integrity protected, but the callback request/response exchange that returns the actual
1156 message MAY be mutually authenticated and integrity protected, depending on the environment of use.

1157 If the actual SAML protocol message is intended for a specific recipient, then the artifact's issuer MUST
1158 authenticate the sender of the subsequent `<samlp:ArtifactResolve>` message before returning the
1159 actual message.

1160 The transmission of an artifact to and from the user agent SHOULD be protected with confidentiality; SSL
1161 3.0 [SSL3] or TLS 1.0 [RFC2246] SHOULD be used. The callback request/response exchange that
1162 returns the actual message MAY be protected, depending on the environment of use.

1163 In general, this binding relies on the artifact as a hard-to-forge short-term reference and applies other
1164 security measures to the callback request/response that returns the actual message. All artifacts MUST
1165 have a single-use semantic enforced by the artifact issuer.

1166 Furthermore, it is RECOMMENDED that artifact receivers also enforce a single-use semantic on the
1167 artifact values they receive, to prevent an attacker from interfering with the resolution of an artifact by a
1168 user agent and then resubmitting it to the artifact receiver. If an attempt to resolve an artifact does not
1169 complete successfully, the artifact SHOULD be placed into a blocked artifact list for a period of time that
1170 exceeds a reasonable acceptance period during which the artifact issuer would resolve the artifact.

1171 Note also that there is no mechanism defined to protect the integrity of the relationship between the
1172 artifact and the "RelayState" value, if any. That is, an attacker can potentially recombine a pair of valid
1173 HTTP responses by switching the "RelayState" values associated with each artifact. As a result, the
1174 producer/consumer of "RelayState" information MUST take care not to associate sensitive state
1175 information with the "RelayState" value without taking additional precautions (such as based on the
1176 information in the SAML protocol message retrieved via artifact).

1177 **3.6.6 Error Reporting**

1178 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD
1179 return a response message with a second-level `<samlp:StatusCode>` value of
1180 `urn:oasis:names:tc:SAML:2.0:status:RequestDenied`.

1181 HTTP interactions during the message exchange MUST NOT use HTTP error status codes to indicate
1182 failures in SAML processing, since the user agent is not a full party to the SAML protocol exchange.

1183 If the issuer of an artifact receives a `<samlp:ArtifactResolve>` message that it can understand, it
1184 MUST return a `<samlp:ArtifactResponse>` with a `<samlp:StatusCode>` value of
1185 `urn:oasis:names:tc:SAML:2.0:status:Success`, even if it does not return the corresponding
1186 message (for example because the artifact requester is not authorized to receive the message or the
1187 artifact is no longer valid).

1188 For more information about SAML status codes, see the SAML assertions and protocols specification
1189 [SAMLCore].

1190 **3.6.7 Metadata Considerations**

1191 Support for the HTTP Artifact binding SHOULD be reflected by indicating URL endpoints at which
1192 requests and responses for a particular protocol or profile should be sent. Either a single endpoint or
1193 distinct request and response endpoints MAY be supplied. One or more indexed endpoints for processing
1194 `<samlp:ArtifactResolve>` messages SHOULD also be described.

1195 **3.6.8 Example SAML Message Exchange Using HTTP Artifact**

1196 In this example, a `<LogoutRequest>` and `<LogoutResponse>` message pair are exchanged using the
1197 HTTP Artifact binding, with the artifact resolution taking place using the SOAP binding bound to HTTP.

1198 First, here are the actual SAML protocol messages being exchanged:

```
1199 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
1200 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
1201 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
1202 21T19:00:49Z" Version="2.0">  
1203 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
1204 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
1205 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
1206 <samlp:SessionIndex>1</samlp:SessionIndex>  
1207 </samlp:LogoutRequest>
```

```

1208 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1209 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1210 ID="b0730d21b628110d8b7e004005b13a2b"
1211 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"
1212 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">
1213 <Issuer>https://ServiceProvider.com/SAML</Issuer>
1214 <samlp:Status>
1215 <samlp:StatusCode
1216 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1217 </samlp:Status>
1218 </samlp:LogoutResponse>

```

1219 The initial HTTP request from the user agent in step 1 is not defined by this binding. To initiate the logout
1220 protocol exchange, the SAML requester returns the following HTTP response, containing a SAML artifact.
1221 Note that the line feeds in the HTTP Location header below are a result of document formatting, and
1222 there are no line feeds in the actual header value.

```

1223 HTTP/1.1 302 Object Moved
1224 Date: 21 Jan 2004 07:00:49 GMT
1225 Location:
1226 https://ServiceProvider.com/SAML/SLO/Browser?SAMLart=AAQAADWNEw5VT47wcO4z
1227 X%2FiEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU%
1228 3D&RelayState=0043bfclbc45110dae17004005b13a2b
1229 Content-Type: text/html; charset=iso-8859-1

```

1230 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact
1231 Resolution protocol and the SOAP binding in steps 3 and 4, as follows:

1232 Step 3:

```

1233 POST /SAML/Artifact/Resolve HTTP/1.1
1234 Host: IdentityProvider.com
1235 Content-Type: text/xml
1236 Content-Length: nnn
1237 SOAPAction: http://www.oasis-open.org/committees/security
1238 <SOAP-ENV:Envelope
1239 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1240 <SOAP-ENV:Body>
1241 <samlp:ArtifactResolve
1242 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1243 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1244 ID="_6c3a4f8b9c2d" Version="2.0"
1245 IssueInstant="2004-01-21T19:00:49Z">
1246 <Issuer>https://ServiceProvider.com/SAML</Issuer>
1247 <Artifact>
1248 AAQAADWNEw5VT47wcO4zX/iEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU=
1249 </Artifact>
1250 </samlp:ArtifactResolve>
1251 </SOAP-ENV:Body>
1252 </SOAP-ENV:Envelope>

```

1253 Step 4:

```

1254 HTTP/1.1 200 OK
1255 Date: 21 Jan 2004 07:00:49 GMT
1256 Content-Type: text/xml
1257 Content-Length: nnnn
1258 <SOAP-ENV:Envelope
1259 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1260 <SOAP-ENV:Body>
1261 <samlp:ArtifactResponse
1262 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1263 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1264 ID="FQvGknDfws2Z" Version="2.0"
1265 InResponseTo="_6c3a4f8b9c2d"
1266 IssueInstant="2004-01-21T19:00:49Z">
1267 <Issuer>https://IdentityProvider.com/SAML</Issuer>
1268 <samlp:Status>

```

```

1269         <samlp:StatusCode
1270         Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1271     </samlp:Status>
1272     <samlp:LogoutRequest ID="d2b7c388cec36fa7c39c28fd298644a8"
1273     IssueInstant="2004-01-21T19:00:49Z"
1274     Version="2.0">
1275         <Issuer>https://IdentityProvider.com/SAML</Issuer>
1276         <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
1277 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>
1278         <samlp:SessionIndex>1</samlp:SessionIndex>
1279     </samlp:LogoutRequest>
1280 </samlp:ArtifactResponse>
1281 </SOAP-ENV:Body>
1282 </SOAP-ENV:Envelope>

```

1283 After any unspecified interactions may have taken place, the SAML responder returns a second SAML
1284 artifact in its HTTP response in step 6:

```

1285 HTTP/1.1 302 Object Moved
1286 Date: 21 Jan 2004 07:05:49 GMT
1287 Location:
1288 https://IdentityProvider.com/SAML/SLO/Response?SAMLart=AAQAAFQIZXv5%
1289 2BQaBaE5qYurHWJO1nAgLASqfnyidHIggbFU0mlSGFTyQiPc%
1290 3D&RelayState=0043bfc1bc45110dae17004005b13a2b
1291 Content-Type: text/html; charset=iso-8859-1

```

1292 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact
1293 Resolution protocol and the SOAP binding in steps 7 and 8, as follows:

1294 Step 7:

```

1295 POST /SAML/Artifact/Resolve HTTP/1.1
1296 Host: ServiceProvider.com
1297 Content-Type: text/xml
1298 Content-Length: nnn
1299 SOAPAction: http://www.oasis-open.org/committees/security
1300 <SOAP-ENV:Envelope
1301   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1302   <SOAP-ENV:Body>
1303     <samlp:ArtifactResolve
1304       xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1305       xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1306       ID="_ec36fa7c39" Version="2.0"
1307       IssueInstant="2004-01-21T19:05:49Z">
1308         <Issuer>https://IdentityProvider.com/SAML</Issuer>
1309         <Artifact>
1310           AAQAAFQIZXv5+QaBaE5qYurHWJO1nAgLASqfnyidHIggbFU0mlSGFTyQiPc=
1311         </Artifact>
1312       </samlp:ArtifactResolve>
1313     </SOAP-ENV:Body>
1314   </SOAP-ENV:Envelope>

```

1315 Step 8:

```

1316 HTTP/1.1 200 OK
1317 Date: 21 Jan 2004 07:05:49 GMT
1318 Content-Type: text/xml
1319 Content-Length: nnnn
1320 <SOAP-ENV:Envelope
1321   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1322   <SOAP-ENV:Body>
1323     <samlp:ArtifactResponse
1324       xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1325       xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1326       ID="_FQvGknDfws2Z" Version="2.0"
1327       InResponseTo="_ec36fa7c39"
1328       IssueInstant="2004-01-21T19:05:49Z">
1329     <Issuer>https://ServiceProvider.com/SAML</Issuer>

```

```
1330     <samlp:Status>
1331         <samlp:StatusCode
1332             Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1333     </samlp:Status>
1334     <samlp:LogoutResponse ID="_b0730d21b628110d8b7e004005b13a2b"
1335         InResponseTo="_d2b7c388cec36fa7c39c28fd298644a8"
1336         IssueInstant="2004-01-21T19:05:49Z"
1337         Version="2.0">
1338         <Issuer>https://ServiceProvider.com/SAML</Issuer>
1339         <samlp:Status>
1340             <samlp:StatusCode
1341                 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1342             </samlp:Status>
1343         </samlp:LogoutResponse>
1344     </samlp:ArtifactResponse>
1345 </SOAP-ENV:Body>
1346 </SOAP-ENV:Envelope>
```

1347 3.7 SAML URI Binding

1348 URIs are a protocol-independent means of referring to a resource. This binding is not a general SAML
1349 request/response binding, but rather supports the encapsulation of a `<samlp:AssertionIDRequest>`
1350 message with a single `<saml:AssertionIDRef>` into the resolution of a URI. The result of a successful
1351 request is a SAML `<saml:Assertion>` element (but not a complete SAML response).

1352 Like SOAP, URI resolution can occur over multiple underlying transports. This binding has transport-
1353 independent aspects, but also calls out the use of HTTP with SSL 3.0 [SSL3] or TLS 1.0 [RFC2246] as
1354 REQUIRED (mandatory to implement).

1355 3.7.1 Required Information

1356 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:URI

1357 **Contact information:** security-services-comment@lists.oasis-open.org

1358 **Description:** Given below.

1359 **Updates:** None

1360 3.7.2 Protocol-Independent Aspects of the SAML URI Binding

1361 The following sections define aspects of the SAML URI binding that are independent of the underlying
1362 transport protocol of the URI resolution process.

1363 3.7.2.1 Basic Operation

1364 A SAML URI reference identifies a specific SAML assertion. The result of resolving the URI MUST be a
1365 message containing the assertion, or a transport-specific error. The specific format of the message
1366 depends on the underlying transport protocol. If the transport protocol permits the returned content to be
1367 described, such as HTTP 1.1 [RFC2616], then the assertion MAY be encoded in whatever format is
1368 permitted. If not, the assertion MUST be returned in a form which can be unambiguously interpreted as or
1369 transformed into an XML serialization of the assertion.

1370 It MUST be the case that if the same URI reference is resolved in the future, then either the same SAML
1371 assertion, or an error, is returned. That is, the reference MAY be persistent but MUST consistently
1372 reference the same assertion, if any.

1373 **3.7.3 Security Considerations**

1374 Indirect use of a SAML assertion presents dangers if the binding of the reference to the result is not
1375 secure. The particular threats and their severity depend on the use to which the assertion is being put. In
1376 general, the result of resolving a URI reference to a SAML assertion SHOULD only be trusted if the
1377 requester can be certain of the identity of the responder and that the contents have not been modified in
1378 transit.

1379 It is often not sufficient that the assertion itself be signed, because URI references are by their nature
1380 somewhat opaque to the requester. The requester SHOULD have independent means to ensure that the
1381 assertion returned is actually the one that is represented by the URI; this is accomplished by both
1382 authenticating the responder and relying on the integrity of the response.

1383 **3.7.4 MIME Encapsulation**

1384 For resolution protocols that support MIME as a content description and packaging mechanism, the
1385 resulting assertion SHOULD be returned as a MIME entity of type `application/samlassertion+xml`,
1386 as defined by [SAMLmime].

1387 **3.7.5 Use of HTTP URIs**

1388 A SAML authority that claims conformance to the SAML URI binding MUST implement support for HTTP.
1389 This section describes certain specifics of using HTTP URIs, including URI syntax, HTTP headers, and
1390 error reporting.

1391 **3.7.5.1 URI Syntax**

1392 In general, there are no restrictions on the permissible syntax of a SAML URI reference as long as the
1393 SAML authority responsible for the reference creates the message containing it. However, authorities
1394 MUST support a URL endpoint at which an HTTP request can be sent with a single query string
1395 parameter named `ID`. There MUST be no query string in the endpoint URL itself independent of this
1396 parameter.

1397 For example, if the documented endpoint at an authority is "https://saml.example.edu/assertions", a
1398 request for an assertion with an `ID` of `abcde` can be sent to:

1399 `https://saml.example.edu/assertions?ID=abcde`

1400 Note that the use of wildcards is not allowed for such ID queries.

1401 **3.7.5.2 HTTP and Caching Considerations**

1402 HTTP proxies MUST NOT cache SAML assertions. To ensure this, the following rules SHOULD be
1403 followed.

1404 When returning SAML assertions using HTTP 1.1, HTTP responders SHOULD:

- 1405 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 1406 • Include a `Pragma` header field set to "no-cache".

1407 **3.7.5.3 Security Considerations**

1408 RFC 2617 [RFC2617] describes possible attacks in the HTTP environment when basic or message-digest
1409 authentication schemes are used.

1410 Use of SSL 3.0 [SSL3] or TLS 1.0 [RFC2246] is STRONGLY RECOMMENDED as a means of
1411 authentication, integrity protection, and confidentiality.

1412 **3.7.5.4 Error Reporting**

1413 As an HTTP protocol exchange, the appropriate HTTP status code SHOULD be used to indicate the result
1414 of a request. For example, a SAML responder that refuses to perform a message exchange with the
1415 SAML requester SHOULD return a "403 Forbidden" response. If the assertion specified is unknown to
1416 the responder, then a "404 Not Found" response SHOULD be returned. In these cases, the content of
1417 the HTTP body is not significant.

1418 **3.7.5.5 Metadata Considerations**

1419 Support for the URI binding over HTTP SHOULD be reflected by indicating a URL endpoint at which
1420 requests for arbitrary assertions are to be sent.

1421 **3.7.5.6 Example SAML Message Exchange Using an HTTP URI**

1422 Following is an example of a request for an assertion.

```
1423 GET /SamlService?ID=abcde HTTP/1.1  
1424 Host: www.example.com
```

1425 Following is an example of the corresponding response, which supplies the requested assertion.

```
1426 HTTP/1.1 200 OK  
1427 Content-Type: application/samlassertion+xml  
1428 Cache-Control: no-cache, no-store  
1429 Pragma: no-cache  
1430 Content-Length: nnnn  
  
1431 <saml:Assertion ID="abcde" ...>  
1432 ...  
1433 </saml:Assertion>
```

4 References

1434

- 1435 **[HTML401]** HTML 4.01 Specification, W3C Recommendation 24 December 1999,
1436 <http://www.w3.org/TR/html4>.
- 1437 **[Liberty]** The Liberty Alliance Project, <http://www.projectliberty.org>.
- 1438 **[PAOS]** Aarts, R., "Liberty Reverse HTTP Binding for SOAP Specification", Version: 1.0,
1439 <https://www.projectliberty.org/specs/liberty-paos-v1.0.pdf>
- 1440 **[RFC1750]** Randomness Recommendations for Security. <http://www.ietf.org/rfc/rfc1750.txt>
- 1441 **[RFC2045]** Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet
1442 Message Bodies, <http://www.ietf.org/rfc/rfc2045.txt>
- 1443 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF
1444 RFC 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>.
- 1445 **[RFC2246]** The TLS Protocol Version 1.0, <http://www.ietf.org/rfc/rfc2246.txt>.
- 1446 **[RFC2279]** UTF-8, a transformation format of ISO 10646, <http://www.ietf.org/rfc/rfc2279.txt>.
- 1447 **[RFC2616]** Hypertext Transfer Protocol -- HTTP/1.1, <http://www.ietf.org/rfc/rfc2616.txt>.
- 1448 **[RFC2617]** *HTTP Authentication: Basic and Digest Access Authentication*, IETF RFC 2617,
1449 <http://www.ietf.org/rfc/rfc2617.txt>.
- 1450 **[SAML11Bind]** E. Maler et al. *Bindings and Profiles for the OASIS Security Assertion Markup
1451 Language (SAML)*. OASIS, September 2003. Document ID oasis-sstc-saml-
1452 bindings-1.1. <http://www.oasis-open.org/committees/security/>.
- 1453 **[SAMLConform]** P. Mishra et al. *Conformance Requirements for the OASIS Security Assertion
1454 Markup Language (SAML) V2.0*. OASIS SSTC, January 2005. Document ID sstc-
1455 saml-conformance-2.0-cd-04. <http://www.oasis-open.org/committees/security/>.
- 1456 **[SAMLCore]** S. Cantor et al., *Assertions and Protocols for the OASIS Security Assertion
1457 Markup Language (SAML) V2.0*. OASIS SSTC, December 2004/January 2005.
1458 Document ID sstc-saml-core-2.0-cd-04. See [http://www.oasis-
open.org/committees/security/](http://www.oasis-
1459 open.org/committees/security/).
- 1460 **[SAMLGloss]** J. Hodges et al., *Glossary for the OASIS Security Assertion Markup Language
1461 (SAML) V2.0*. OASIS SSTC, January 2005. Document ID sstc-saml-glossary-2.0-
1462 cd-04. See <http://www.oasis-open.org/committees/security/>.
- 1463 **[SAMLMeta]** S. Cantor et al., *Metadata for the OASIS Security Assertion Markup Language
1464 (SAML) V2.0*. OASIS SSTC, January 2005. Document ID sstc-saml-metadata-
1465 2.0-cd-04. See <http://www.oasis-open.org/committees/security/>.
- 1466 **[SAMLmime]** application/saml+xml Media Type Registration, IETF Internet-Draft,
1467 <http://www.ietf.org/internet-drafts/draft-hodges-saml-mediatype-01.txt>.
- 1468 **[SAMLProfile]** S. Cantor et al., *Profiles for the OASIS Security Assertion Markup Language
1469 (SAML) V2.0*. OASIS SSTC, January 2005. Document ID sstc-saml-profiles-2.0-
1470 cd-04. See <http://www.oasis-open.org/committees/security/>.
- 1471 **[SAMLSecure]** F. Hirsch et al., *Security and Privacy Considerations for the OASIS Security
1472 Assertion Markup Language (SAML) V2.0*. OASIS SSTC, January 2005.
1473 Document ID sstc-saml-sec-consider-2.0-cd-04. See [http://www.oasis-
open.org/committees/security/](http://www.oasis-
1474 open.org/committees/security/).
- 1475 **[SOAP11]** D. Box et al., *Simple Object Access Protocol (SOAP) 1.1*, World Wide Web
1476 Consortium Note, May 2000, [http://www.w3.org/TR/2000/NOTE-SOAP-
20000508/](http://www.w3.org/TR/2000/NOTE-SOAP-
1477 20000508/).
- 1478 **[SOAP-PRIMER]** N. Mitra, SOAP Version 1.2 Part 0: Primer, W3C Recommendation 24 June
1479 2003, <http://www.w3.org/TR/soap12-part0/>

1480	[SSL3]	A. Frier et al., <i>The SSL 3.0 Protocol</i> , Netscape Communications Corp, November 1996.
1481		
1482	[SSTCWeb]	OASIS Security Services Technical Committee website, http://www.oasis-open.org/committees/security .
1483		
1484	[XHTML]	XHTML 1.0 The Extensible HyperText Markup Language (Second Edition), http://www.w3.org/TR/xhtml1/ .
1485		
1486	[XMLSig]	D. Eastlake et al., <i>XML-Signature Syntax and Processing</i> , World Wide Web Consortium, http://www.w3.org/TR/xmlsig-core/ .
1487		

1488 **Appendix A. Registration of MIME media type**
1489 **application/samlassertion+xml**

1490 **Introduction**

1491 The IESG has approved a request to register the `application/samlassertion+xml` MIME
1492 media type in the standards tree. This media type is a product of the Organization for the
1493 Advancement of Structured Information Systems (OASIS).

1494 The IESG contact persons are Ted Hardie and Scott Hollenbeck.

1495 **MIME media type name**

1496 `application`

1497 **MIME subtype name**

1498 `samlassertion+xml`

1499 **Required parameters**

1500 None

1501 **Optional parameters**

1502 `charset`

1503 Same as `charset` parameter of `application/xml` [RFC3023].

1504 **Encoding considerations**

1505 Same as for `application/xml` [RFC3023].

1506 **Security considerations**

1507 Per their specification, `samlassertion+xml`-typed objects do not contain executable content.
1508 However, SAML assertions are XML-based objects [XML]. As such, they have all of the general
1509 security considerations presented in section 10 of [RFC3023], as well as additional ones, since
1510 they are explicit security objects. For example, `samlassertion+xml`-typed objects will often
1511 contain data that may identify or pertain to a natural person, and may be used as a basis for
1512 sessions and access control decisions.

1513 To counter potential issues, `samlassertion+xml`-typed objects contain data that should be
1514 signed appropriately by the sender. Any such signature must be verified by the recipient of the
1515 data - both as a valid signature, and as being the signature of the sender. Issuers of
1516 `samlassertion+xml`-typed objects containing SAMLv2 assertions may also encrypt all, or
1517 portions of, the assertions (see [SAMLv2Core]).

1518 In addition, SAML profiles and protocol bindings specify use of secure channels as appropriate.

1519 [SAMLv2.0] incorporates various privacy-protection techniques in its design. For example: opaque
1520 handles, specific to interactions between specific system entities, may be assigned to subjects.
1521 The handles are mappable to wider-context identifiers (e.g. email addresses, account identifiers,
1522 etc) by only the specific parties.

1523 For a more detailed discussion of SAML security considerations and specific security-related
1524 design techniques, please refer to the SAML specifications listed in the below bibliography. The
1525 specifications containing security-specific information have been explicitly listed for each version
1526 of SAML.

1527 **Interoperability considerations**

1528 SAML assertions are explicitly versioned. Relying parties should ensure that they observe
1529 assertion version information and behave accordingly. See chapters on SAML Versioning in
1530 [SAMLv1Core], [SAMLv11Core], or [SAMLv2Core], as appropriate.

1531 **Published specification**

1532 [SAMLv2Bind] explicitly specifies use of the `application/samlassertion+xml` MIME media
1533 type. However, it is conceivable that non-SAMLv2 assertions (i.e., SAMLv1 and/or SAMLv1.1)
1534 might in practice be conveyed using SAMLv2 bindings.

1535 **Applications which use this media type**

1536 Potentially any application implementing SAML, as well as those applications implementing
1537 specifications based on SAML, e.g. those available from the Liberty Alliance [LAP].

1538 **Additional information**

1539 **Magic number(s)**

1540 In general, the same as for `application/xml` [RFC3023]. In particular, the XML root element of the
1541 returned object will be `<saml:Assertion>`, where "saml" maps to a version-specific SAML
1542 assertion namespace, as defined by the appropriate SAML "core" specification (see bibliography).
1543 In the case of SAMLv2.0, the root element of the returned object may be either
1544 `<saml:Assertion>` or `<saml:EncryptedAssertion>`, where "saml" maps to the SAMLv2.0
1545 assertion namespace URI:

1546 `urn:oasis:names:tc:SAML:2.0:assertion`

1547 **File extension(s)**

1548 None

1549 **Macintosh File Type Code(s)**

1550 None

1551 **Person & email address to contact for further information**

1552 This registration is made on behalf of the OASIS Security Services Technical Committee (SSTC)
1553 Please refer to the SSTC website for current information on committee chairperson(s) and their
1554 contact addresses: <http://www.oasis-open.org/committees/security/>. Committee members should
1555 submit comments and potential errata to the security-services@lists.oasis-open.org list. Others
1556 should submit them by filling out the web form located at [http://www.oasis-
1557 open.org/committees/comments/form.php?wg_abbrev=security](http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=security).

1558 Additionally, the SAML developer community email distribution list, [saml-dev@lists.oasis-
1559 open.org](mailto:saml-dev@lists.oasis-open.org), may be employed to discuss usage of the `application/samlassertion+xml` MIME media
1560 type. The "saml-dev" mailing list is publicly archived here: [http://lists.oasis-
1561 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

1562 subscribe, send a message with the single word "subscribe" in the message body, to: [saml-dev-](mailto:saml-dev-request@lists.oasis-open.org)
1563 [request@lists.oasis-open.org](mailto:saml-dev-request@lists.oasis-open.org).

1564 Intended usage

1565 COMMON

1566 Author/Change controller

1567 The SAML specification sets are a work product of the OASIS Security Services Technical
1568 Committee (SSTC). OASIS and the SSTC have change control over the SAML specification sets.

1569 Bibliography

- 1570 [LAP] *"Liberty Alliance Project"*. See <http://www.projectliberty.org/>
- 1571 [OASIS] *"Organization for the Advancement of Structured Information Systems"*.
1572 See <http://www.oasis-open.org/>
- 1573 [RFC3023] M. Murata, S. St.Laurent, D. Kohn, *"XML Media Types"*, IETF Request for
1574 Comments 3023, January 2001. Available as [http://www.rfc-](http://www.rfc-editor.org/rfc/rfc3023.txt)
1575 [editor.org/rfc/rfc3023.txt](http://www.rfc-editor.org/rfc/rfc3023.txt)
- 1576 [SAMLv1.0] OASIS Security Services Technical Committee, *"Security Assertion*
1577 *Markup Language (SAML) Version 1.0 Specification Set"*. OASIS
1578 Standard 200205, November 2002. Available as [http://www.oasis-](http://www.oasis-open.org/committees/download.php/2290/oasis-sstc-saml-1.0.zip)
1579 [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)
- 1580 [SAMLv1Bind] Prateek Mishra et al., *"Bindings and Profiles for the OASIS Security*
1581 *Assertion Markup Language (SAML)"*, OASIS, November 2002.
1582 Document ID oasis-sstc-saml-bindings-1.0. See [http://www.oasis-](http://www.oasis-open.org/committees/security/)
1583 [open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1584 [SAMLv1Core] Phillip Hallam-Baker et al., *"Assertions and Protocol for the OASIS*
1585 *Security Assertion Markup Language (SAML)"*, OASIS, November 2002.
1586 Document ID oasis-sstc-saml-core-1.0. See [http://www.oasis-](http://www.oasis-open.org/committees/security/)
1587 [open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1588 [SAMLv1Sec] Chris McLaren et al., *"Security Considerations for the OASIS Security*
1589 *Assertion Markup Language (SAML)"*, OASIS, November 2002.
1590 Document ID oasis-sstc-saml-sec-consider-1.0. See [http://www.oasis-](http://www.oasis-open.org/committees/security/)
1591 [open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1592 [SAMLv1.1] OASIS Security Services Technical Committee, *"Security Assertion*
1593 *Markup Language (SAML) Version 1.1 Specification Set"*. OASIS
1594 Standard 200308, August 2003. Available as [http://www.oasis-](http://www.oasis-open.org/committees/download.php/3400/oasis-sstc-saml-1.1-pdf-xsd.zip)
1595 [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)
- 1596 [SAMLv11Bind] E. Maler et al. *"Bindings and Profiles for the OASIS Security Assertion*
1597 *Markup Language (SAML)"*. OASIS, September 2003. Document ID
1598 oasis-sstc-saml-bindings-1.1. [http://www.oasis-](http://www.oasis-open.org/committees/security/)
1599 [open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1600 [SAMLv11Core] E. Maler et al. *"Assertions and Protocol for the OASIS Security Assertion*
1601 *Markup Language (SAML)"*. OASIS, September 2003. Document ID
1602 oasis-sstc-saml-core-1.1. <http://www.oasis-open.org/committees/security/>
- 1603 [SAMLv11Sec] E. Maler et al. *"Security Considerations for the OASIS Security Assertion*
1604 *Markup Language (SAML)"*. OASIS, September 2003. Document ID
1605 oasis-sstc-saml-sec-consider-1.1. [http://www.oasis-](http://www.oasis-open.org/committees/security/)
1606 [open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1607 [SAMLv2.0] OASIS Security Services Technical Committee, *"Security Assertion*
1608 *Markup Language (SAML) Version 2.0 Specification Set"*. WORK IN
1609 PROGRESS. Available at <http://www.oasis->

1610		open.org/committees/security/
1611	[SAMLv2Bind]	S. Cantor et al., " <i>Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0</i> ". OASIS SSTC, August 2004. Document ID sstc-saml-bindings-2.0-cd-03, WORK IN PROGRESS. See http://www.oasis-open.org/committees/security/
1612		
1613		
1614		
1615	[SAMLv2Core]	S. Cantor et al., " <i>Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0</i> ". OASIS SSTC, August 2004. Document ID sstc-saml-core-2.0-cd-03, WORK IN PROGRESS. See http://www.oasis-open.org/committees/security/
1616		
1617		
1618		
1619	[SAMLv2Prof]	S. Cantor et al., " <i>Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0</i> ". OASIS SSTC, August 2004. Document ID sstc-saml-profiles-2.0-cd-03, WORK IN PROGRESS. See http://www.oasis-open.org/committees/security/
1620		
1621		
1622		
1623	[SAMLv2Sec]	F. Hirsch et al., " <i>Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0</i> ". OASIS SSTC, August 2004, WORK IN PROGRESS. Document ID sstc-saml-sec-consider-2.0-cd-03. See http://www.oasis-open.org/committees/security/
1624		
1625		
1626		
1627	[SSTC]	"OASIS Security Services Technical Committee". See http://www.oasis-open.org/committees/security/
1628		
1629	[XML]	Bray, T., Paoli, J., Sperberg-McQueen, C.M. and E. Maler, François Yergeau, " <i>Extensible Markup Language (XML) 1.0 (Third Edition)</i> ", World Wide Web Consortium Recommendation REC-xml, Feb 2004, Available as http://www.w3.org/TR/REC-xml/
1630		
1631		
1632		

Appendix B. Acknowledgments

1634 The editors would like to acknowledge the contributions of the OASIS Security Services Technical
1635 Committee, whose voting members at the time of publication were:

- 1636 • Conor Cahill, AOL
- 1637 • John Hughes, Atos Origin
- 1638 • Hal Lockhart, BEA Systems
- 1639 • Mike Beach, Boeing
- 1640 • Rebekah Metz, Booz Allen Hamilton
- 1641 • Rick Randall, Booz Allen Hamilton
- 1642 • Ronald Jacobson, Computer Associates
- 1643 • Carolina Canales-Valenzuela, Ericsson
- 1644 • Dana Kaufman, Forum Systems
- 1645 • Irving Reid, Hewlett-Packard
- 1646 • Paula Austel, IBM
- 1647 • Michael McIntosh, IBM
- 1648 • Anthony Nadalin, IBM
- 1649 • Nick Ragouzis, Individual
- 1650 • Scott Cantor, Internet2
- 1651 • Bob Morgan, Internet2
- 1652 • Peter Davis, Neustar
- 1653 • Jeff Hodges, Neustar
- 1654 • Frederick Hirsch, Nokia
- 1655 • Senthil Sengodan, Nokia
- 1656 • Abbie Barbir, Nortel Networks
- 1657 • Scott Kiestler, Novell
- 1658 • Cameron Morris, Novell
- 1659 • Paul Madsen, NTT
- 1660 • Steve Anderson, OpenNetwork
- 1661 • Ari Kermaier, Oracle
- 1662 • Vamsi Motukuru, Oracle
- 1663 • Darren Platt, Ping Identity
- 1664 • Prateek Mishra, Principal Identity
- 1665 • Jim Lien, RSA Security
- 1666 • John Linn, RSA Security
- 1667 • Rob Philpott, RSA Security
- 1668 • Dipak Chopra, SAP
- 1669 • Jahan Moreh, Sigaba
- 1670 • Bhavna Bhatnagar, Sun Microsystems
- 1671 • Eve Maler, Sun Microsystems
- 1672 • Ronald Monzillo, Sun Microsystems
- 1673 • Emily Xu, Sun Microsystems
- 1674 • Greg Whitehead, Trustgenix

1675 The editors also would like to acknowledge the following people for their contributions to previous versions
1676 of the OASIS Security Assertions Markup Language Standard:

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

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

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

1706 **Appendix C. Notices**

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

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

1718 **Copyright © OASIS Open 2005. All Rights Reserved.**

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

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

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