



---

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

Committee Draft 01, 18 August 2004

**Document identifier:**

sstc-saml-bindings-2.0-cd-01

**Location:**

[http://www.oasis-open.org/committees/documents.php?wg\\_abbrev=security](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  
Hal Lockhart, BEA Systems  
Michael Beach, Boeing  
Rick Randall, Booze, Allen, Hamilton  
Tim Alsop, Cybersafe  
Nick Ragouzis, Enosis  
John Hughes, Entegriy Solutions  
Paul Madsen, Entrust  
Irving Reid, Hewlett-Packard  
Paula Austel, IBM  
Maryann Hondo, IBM  
Michael McIntosh, IBM  
Tony Nadalin, IBM  
Scott Cantor, Internet2  
RL 'Bob' Morgan, Internet2  
Rebekah Metz, NASA  
Prateek Mishra, Netegrity  
Peter C Davis, Neustar  
Frederick Hirsch, Nokia  
John Kemp, Nokia  
Charles Knouse, Oblix  
Steve Anderson, OpenNetwork  
John Linn, RSA Security  
Rob Philpott, RSA Security  
Jahan Moreh, Sigaba  
Anne Anderson, Sun Microsystems  
Jeff Hodges, Sun Microsystems  
Eve Maler, Sun Microsystems

45 Ron Monzillo, Sun Microsystems  
46 Greg Whitehead, Trustgenix

47 **Abstract:**

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

50 **Status:**

51 This is a **Committee Draft** approved by the Security Services Technical Committee on 17 August  
52 2004.

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

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

## 62 Table of Contents

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

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

---

# 1 Introduction

150

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

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

## 1.1 Protocol Binding Concepts

156

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

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

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

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

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

## 1.2 Notation

171

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

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

176 `Example code listings appear like this.`

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

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

Prefix	XML Namespace	Comments
saml:	urn:oasis:names:tc:SAML:2.0:assertion	This is the SAML V2.0 assertion namespace [SAMLCore].
samlp:	urn:oasis:names:tc:SAML:2.0:protocol	This is the SAML V2.0 protocol namespace [SAMLCore].
ds:	http://www.w3.org/2000/09/xmldsig#	This namespace is defined in the XML Signature Syntax and Processing specification [XMLSig] and its governing schema.

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

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

---

## 2 Guidelines for Specifying Additional Protocol Bindings

183  
184

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

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

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

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

---

## 216 3 Protocol Bindings

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

### 218 3.1 General Considerations

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

#### 220 3.1.1 Use of RelayState

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

#### 228 3.1.2 Security

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

##### 231 3.1.2.1 Use of SSL 3.0 or TLS 1.0

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

235 TLS-capable implementations MUST implement the TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA cipher  
236 suite and MAY implement the TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA cipher suite [AES].

237 FIPS TLS-capable implementations MUST implement the corresponding  
238 TLS\_RSA\_FIPS\_WITH\_3DES\_EDE\_CBC\_SHA cipher suite and MAY implement the corresponding  
239 TLS\_RSA\_FIPS\_AES\_128\_CBC\_SHA cipher suite [AES] [FIPS].

240 SSL-capable implementations MUST implement the SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA cipher  
241 suite.

242 FIPS SSL-capable implementations MUST implement the FIPS cipher suite corresponding to the SSL  
243 SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA cipher suite [FIPS].

##### 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 [RFC2617] describes possible attacks in the HTTP environment when basic or message-digest  
272 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 [SOAP1.1]. 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](mailto: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 [SOAP1.1] §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 [SOAP1.1] 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 [SOAP1.1] 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 [SOAP1.1] §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 `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 insure 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 [SOAP1.1] § 6.2, in the case of a SOAP error while processing a SOAP request, the  
376 SOAP 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
418 <SOAP-ENV:Envelope
419   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
420   <SOAP-ENV:Body>
421     <samlp:Response xmlns:samlp="..." xmlns:saml="..." xmlns:ds="..."
422     ID="_6c3a4f8b9c2d" Version="2.0" IssueInstant="2004-03-27T08:42:00Z">
423       <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](mailto: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.

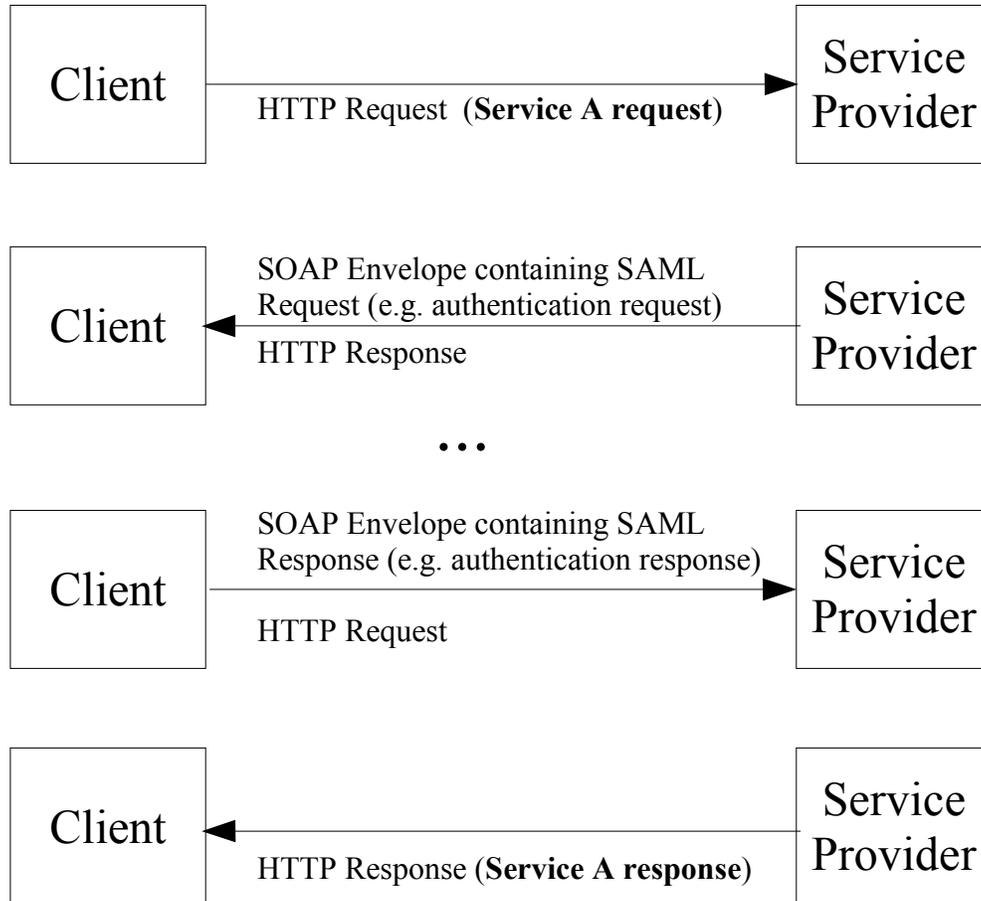


Figure 1: PAOS Binding Message Exchanges

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 insure 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](mailto: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 [RFC2616]) as an intermediary. This  
541 may be necessary, for example, if the communicating parties do not share a direct path of communication.  
542 It may also be needed if the responder requires an interaction with the user agent in order to fulfill the  
543 request, 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 If a SAML request message is accompanied by RelayState data, then the SAML responder MUST return  
552 its SAML protocol response using a binding that also supports a RelayState mechanism, and it MUST  
553 place the exact data it received with the request into the corresponding RelayState parameter in the  
554 response.

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

### 558 **3.4.4 Message Encoding**

559 Messages are encoded for use with this binding using a URL encoding technique, and transmitted using  
560 the HTTP GET method. There are many possible ways to encode XML into a URL, depending on the

561 constraints in effect. This specification defines one such method without precluding others. Binding  
562 endpoints SHOULD indicate which encodings they support using metadata, when appropriate. Particular  
563 encodings MUST be uniquely identified with a URI when defined. It is not a requirement that all possible  
564 SAML messages be encodable with a particular set of rules, but the rules MUST clearly indicate which  
565 messages or content can or cannot be so encoded.

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

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

### 571 3.4.4.1 DEFLATE Encoding

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

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

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

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

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

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

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

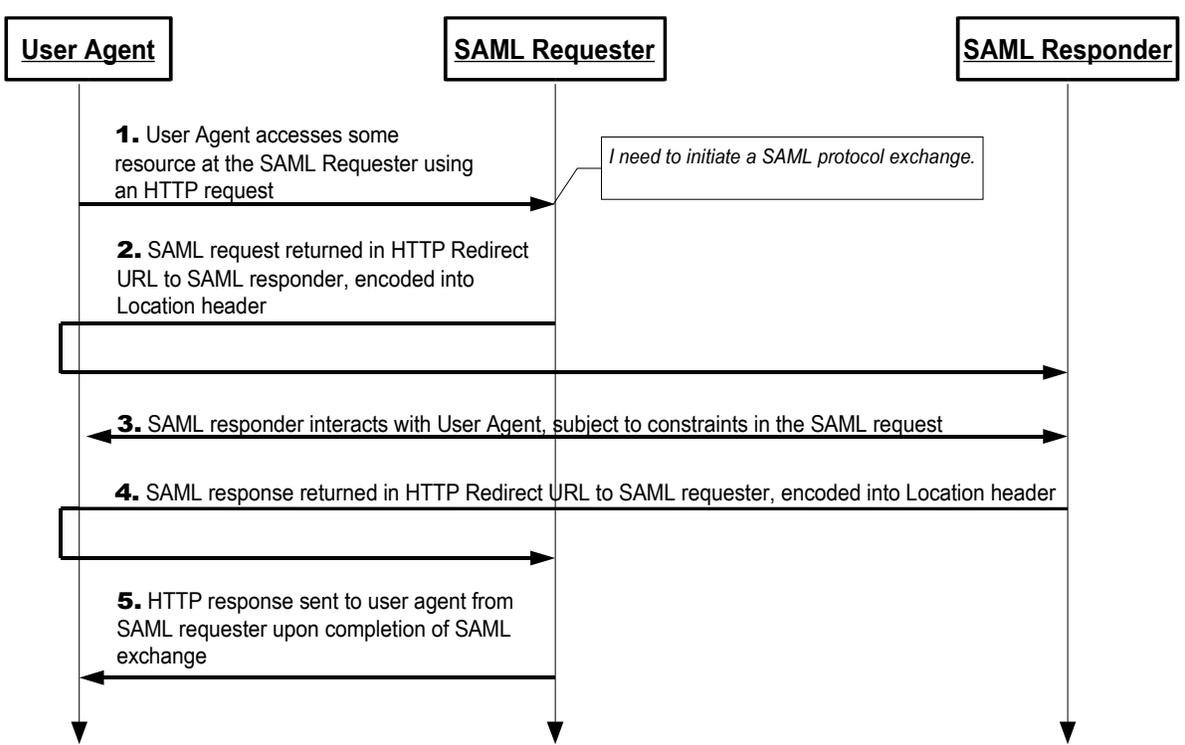
608 in the base64-encoded signature value may themselves require URL-encoding before being added.

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

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

### 613 3.4.5 Message Exchange

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



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

633 SAML requester. The SAML response is returned in the same fashion as described for the SAML  
634 request in step 2.

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

### 637 **3.4.5.1 HTTP and Caching Considerations**

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

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

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

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

### 644 **3.4.5.2 Security Considerations**

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

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

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

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

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

### 661 **3.4.6 Error Reporting**

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

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

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

### 669 **3.4.7 Metadata Considerations**

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

### 673 3.4.8 Example SAML Message Exchange Using HTTP Redirect

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

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

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

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

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

```
703 HTTP/1.1 302 Object Moved  
704 Date: 21 Jan 2004 07:00:49 GMT  
705 Location:  
706 https://ServiceProvider.com/SAML/SLO/Browser?SAMLRequest=H4sIAOCuDUEAA32R  
707 UUvDMBSF3wf9DyXvWZOsq23oCsIQcPuiGZ74liWZVtqk5qYy%2F73puoGCL%2Fhu%2Bfecw%  
708 2B3BNG1Pd%2FYnzv4Z%  
709 2F05aPDxqWsN8HN1hQZnuBXQADei08C95Lv77YazOeG9s95K26Kp5bZYAGjnG2tQNIvDq9crp  
710 NjhTi7yXGq5yI4i%  
711 2FAvJ8qNiRZ6lqchRXAMMujbghfErxAhJMaGY0T0toCE8LV5RvBUf1r1oB2F40ATQmF%  
712 2BAoGpyLM%2FDXPXufQ88SWqljW%  
713 2F895OzX43Sbi5t14z7lslFee17DGHqdfxgXSf87ZQjaRQ%  
714 2BngW8H3cAH2xQRchSkEwTLFTOMKVEYbFcZjhECqUDXQh2KJPJ6mo8XWenYUxSG6VPFS2Tf2g  
715 0u%2BI%2Fpww8mv0ALfRRUQBAAA%  
716 3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F%  
717 2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-  
718 sha1&Signature=NOTAREALSIGNATUREBUTHEREALONEWOULDGOHERE  
719 Content-Type: text/html; charset=iso-8859-1
```

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

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

726 Location:  
727 https://IdentityProvider.com/SAML/SLO/Response?SAMLResponse=H4sIAK03DUEAA  
728 31RTWvDMAy991cE39vYtTY6pimM7VJoYSylh94cR90yEitYTtnPX5a0sDKoTtLT09PXmkzbdH  
729 qHH9iHd6AOHUH03Ta09JjKWe%  
730 2BdRkMlaWdaIB2sLp730y0XXHceAlps2FTymGyIwIcaHZtFg21fclbyVcIrKcqlVELwSpUr4D  
731 z1%  
732 2FKkUiZEli7buNtUBc1bJcmUTpSzYZHk2g59Zqc6VzNQyTY26KhP1sHUUjAs5k4PgnIu5FAeR  
733 ac5lmp1YtDdf6I%2FgaZhn4AxA7f4AnG1GqfWo5TefIXSk47gAf6ktvHm81BX4hcU2%  
734 2F1lwHV%2BJU9V01SKY0NME%2FYNfsILOaJoeHl%2BNRrYuemuBiMXXDvF9i1t8%  
735 2F8jN7AcCjwc4AEAAA%3D%  
736 3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F%  
737 2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-  
738 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE  
739 Content-Type: text/html; charset=iso-8859-1

## 740 3.5 HTTP POST Binding

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

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

### 746 3.5.1 Required Information

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

748 **Contact information:** [security-services-comment@lists.oasis-open.org](mailto:security-services-comment@lists.oasis-open.org)

749 **Description:** Given below.

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

### 751 3.5.2 Overview

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

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

### 760 3.5.3 RelayState

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

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

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

### 771 **3.5.4 Message Encoding**

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

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

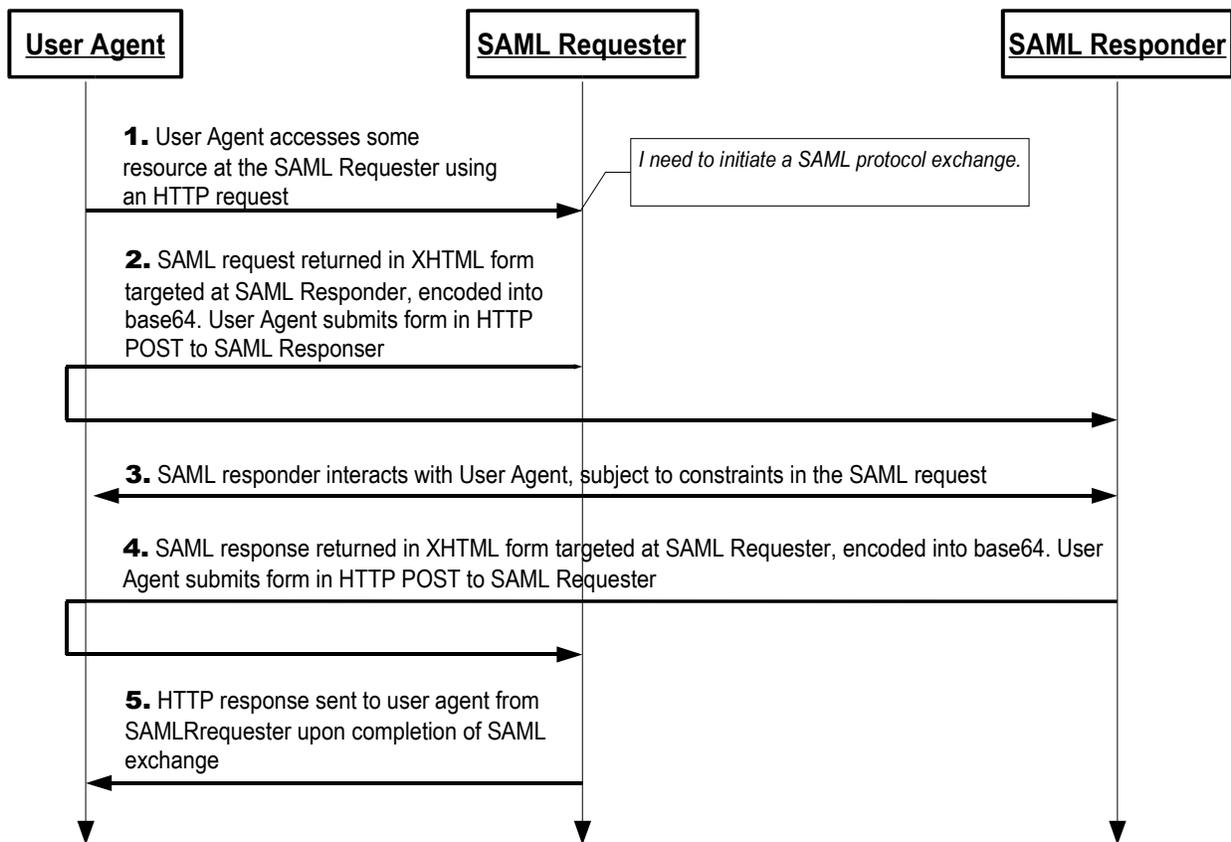
782 If a "RelayState" value is to accompany the SAML protocol message, it MUST be placed in an additional  
783 hidden form control named `RelayState` within the same form with the SAML message.

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

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

### 790 **3.5.5 Message Exchange**

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



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

### 812 3.5.5.1 HTTP and Caching Considerations

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

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

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

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

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

### 819 **3.5.5.2 Security Considerations**

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

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

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

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

### 838 **3.5.6 Error Reporting**

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

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

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

### 846 **3.5.7 Metadata Considerations**

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

### 850 **3.5.8 Example SAML Message Exchange Using HTTP POST**

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

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

```
854 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
855 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
856 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
857 21T19:00:49Z" Version="2.0">  
858 <Issuer>https://IdentityProvider.com/SAML</Issuer>
```





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

## 972 **3.6.2 Overview**

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

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

## 983 **3.6.3 Message Encoding**

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

### 988 **3.6.3.1 RelayState**

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

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

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

### 1000 **3.6.3.2 URL Encoding**

1001 To encode an artifact into a URL, the artifact value is URL-encoded and placed in a query string  
1002 parameter named `SAMLart`.

1003 If a "RelayState" value is to accompany the SAML artifact, it **MUST** be URL-encoded and placed in an  
1004 additional query string parameter named `RelayState`.

### 1005 **3.6.3.3 Form Encoding**

1006 A SAML artifact is form-encoded by placing it in a hidden form control within a form as defined by  
1007 [HTML401], chapter 17. The HTML document **MUST** adhere to the XHTML specification, [XHTML] . The  
1008 form control **MUST** be named `SAMLart`. Any additional form controls or presentation **MAY** be included but  
1009 **MUST NOT** be required in order for the recipient to process the artifact.

1010 If a "RelayState" value is to accompany the SAML artifact, it **MUST** be placed in an additional hidden form  
1011 control named `RelayState`, within the same form with the SAML message.

1012 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using  
1013 this binding to which the artifact is to be delivered. The `method` attribute MUST be set to "POST".

1014 Any technique supported by the user agent MAY be used to cause the submission of the form, and any  
1015 form content necessary to support this MAY be included, such as submit controls and client-side scripting  
1016 commands. However, the recipient MUST be able to process the artifact without regard for the  
1017 mechanism by which the form submission is initiated.

### 1018 3.6.4 Artifact Format

1019 With respect to this binding, an artifact is a short, opaque string. Different types can be defined and used  
1020 without affecting the binding. The important characteristics are the ability of an artifact receiver to identify  
1021 the issuer of the artifact, resistance to tampering and forgery, uniqueness, and compactness.

1022 The general format of any artifact includes a mandatory two-byte artifact type code and a two-byte index  
1023 value identifying a specific endpoint of the artifact resolution service of the issuer, as follows:

```
1024 SAML_artifact      := B64 (TypeCode EndpointIndex RemainingArtifact)
1025 TypeCode           := Byte1Byte2
1026 EndpointIndex     := Byte1Byte2
```

1027 The notation `B64 (TypeCode EndpointIndex RemainingArtifact)` stands for the application of  
1028 the base64 [RFC2045] transformation to the catenation of the `TypeCode`, `EndpointIndex`, and  
1029 `RemainingArtifact`.

1030 The following practices are RECOMMENDED for the creation of SAML artifacts:

- 1031 • Each issuer is assigned an identifying URI, also known as the issuer's entity (or provider) ID. See  
1032 section 8.3.6 of [SAMLCORE] for a discussion of this kind of identifier.
- 1033 • The issuer constructs the `SourceID` component of the artifact by taking the SHA-1 hash of the  
1034 identification URL. The hash value is NOT encoded into hexadecimal.
- 1035 • The `MessageHandle` value is constructed from a cryptographically strong random or  
1036 pseudorandom number sequence [RFC1750] generated by the issuer. The sequence consists of  
1037 values of at least 16 bytes in size. These values should be padded as needed to a total length of 20  
1038 bytes.

1039 The following describes the single artifact type defined by SAML 2.0.

#### 1040 3.6.4.1 Required Information

1041 **Identification:** urn:oasis:names:tc:SAML:2.0:artifact-04

1042 **Contact information:** [security-services-comment@lists.oasis-open.org](mailto:security-services-comment@lists.oasis-open.org)

1043 **Description:** Given below.

1044 **Updates:** None.

#### 1045 3.6.4.2 Format Details

1046 SAML 2.0 defines an artifact type of type code 0x0004. This artifact type is defined as follows:

```
1047 TypeCode           := 0x0004
1048 RemainingArtifact := SourceID MessageHandle
1049 SourceID           := 20-byte_sequence
1050 MessageHandle     := 20-byte_sequence
```

1051 `SourceID` is a 20-byte sequence used by the artifact receiver to determine artifact issuer identity and the  
1052 set of possible resolution endpoints.

1053 It is assumed that the destination site will maintain a table of `SourceID` values as well as one or more  
1054 indexed URL endpoints (or addresses) for the corresponding SAML responder. The SAML metadata  
1055 specification [SAMLMeta] MAY be used for this purpose. On receiving the SAML artifact, the receiver  
1056 determines if the `SourceID` belongs to a known artifact issuer and obtains the location of the SAML  
1057 responder using the `EndpointIndex` before sending a SAML `<samlp:ArtifactResolve>` message  
1058 to it.

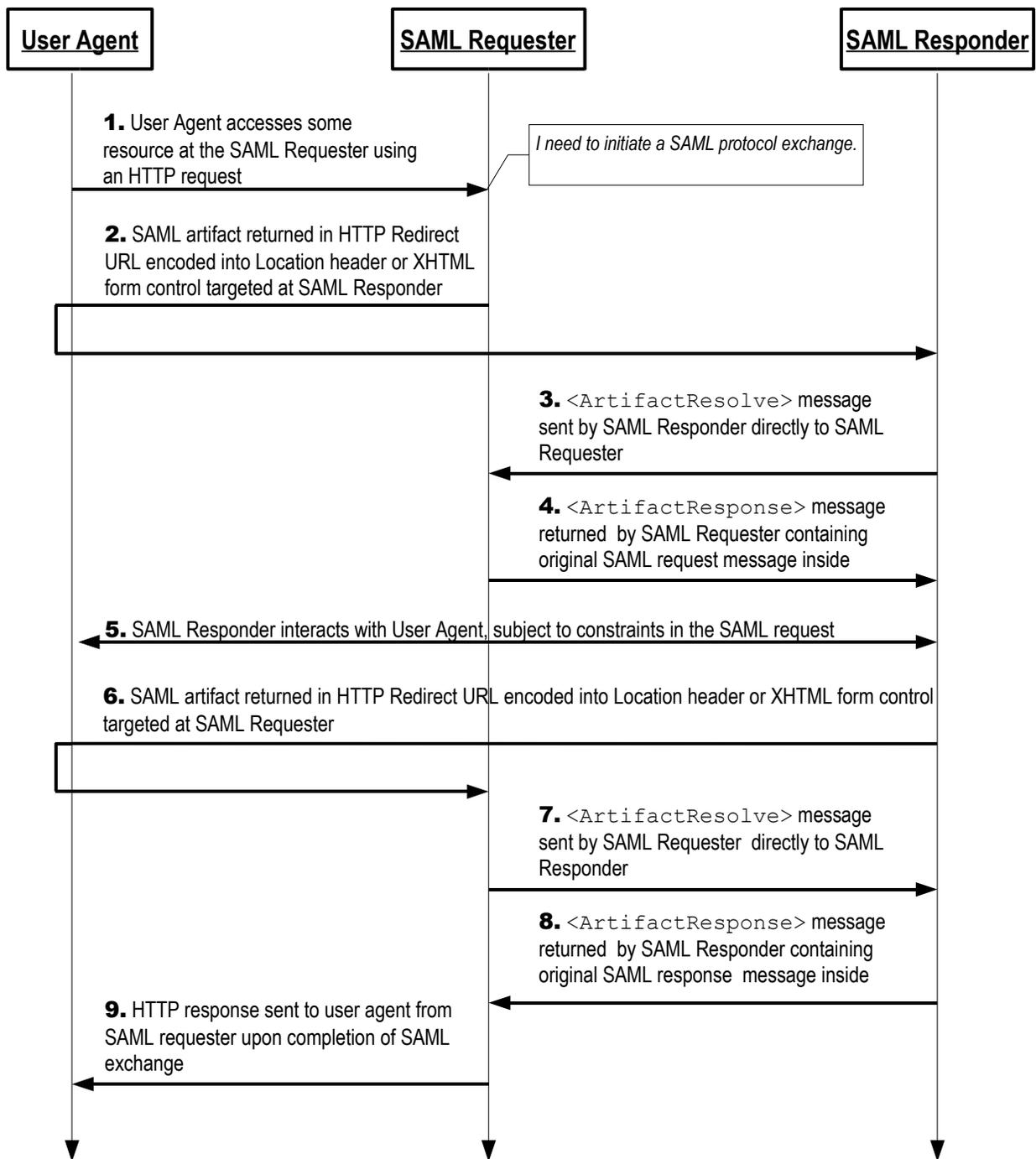
1059 Any two artifact issuers with a common receiver MUST use distinct `SourceID` values. Construction of  
1060 `MessageHandle` values is governed by the principle that they SHOULD have no predictable relationship  
1061 to the contents of the referenced message at the issuing site and it MUST be infeasible to construct or  
1062 guess the value of a valid, outstanding message handle.

### 1063 **3.6.5 Message Exchange**

1064 The system model used for SAML conversations by means of this binding is a request-response model in  
1065 which an artifact reference takes the place of the actual message content, and the artifact reference is  
1066 sent to the user agent in an HTTP response and delivered to the message recipient in an HTTP request.  
1067 The HTTP interactions before, between, and after these exchanges take place is unspecified. Both the  
1068 SAML requester and responder are assumed to be HTTP responders.

1069 Additionally, it is assumed that on receipt of an artifact by way of the user agent, the recipient invokes a  
1070 separate, direct exchange with the artifact issuer using the Artifact Resolution Protocol defined in  
1071 [SAMLCore]. This exchange MUST use a binding that does not use the HTTP user agent as an  
1072 intermediary, such as the SOAP binding. On the successful acquisition of a SAML protocol message, the  
1073 artifact is discarded and the processing of the primary SAML protocol exchange resumes (or ends, if the  
1074 message is a response).

1075 Issuing and delivering an artifact, along with the subsequent resolution step, constitutes half of the overall  
1076 SAML protocol exchange. This binding can be used to deliver either or both halves of a SAML protocol  
1077 exchange. A binding composable with it, such as the HTTP Redirect (see Section 3.4) or POST (see  
1078 Section 3.5) binding, MAY be used to carry the other half of the exchange. The following sequence  
1079 assumes that the artifact binding is used for both halves. See the diagram below illustrating the messages  
1080 exchanged.



- 1081 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of  
1082 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 1083 2. The system entity acting as a SAML requester responds to an HTTP request from the user agent by  
1084 returning an artifact representing a SAML request.
- 1085 • If URL-encoded, the artifact is returned encoded into the HTTP response's Location  
1086 header, and the HTTP status MUST be either 303 or 302. The SAML requester MAY  
1087 include additional presentation and content in the HTTP response to facilitate the user  
1088 agent's transmission of the message, as defined in HTTP 1.1 [RFC2616]. The user

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

### 1114 **3.6.5.1 HTTP and Caching Considerations**

1115 HTTP proxies and the user agent intermediary should not cache SAML artifacts. To insure this, the  
1116 following rules SHOULD be followed.

1117 When returning SAML artifacts using HTTP 1.1, HTTP responders SHOULD:

- 1118 • Include a `Cache-Control` header field set to "no-cache, no-store".  
1119 • Include a `Pragma` header field set to "no-cache".

1120 There are no other restrictions on the use of HTTP headers.

### 1121 **3.6.5.2 Security Considerations**

1122 This binding uses a combination of indirect transmission of a message reference followed by a direct  
1123 exchange to return the actual message. As a result, the message reference (artifact) need not itself be  
1124 authenticated or integrity protected, but the callback request/response exchange that returns the actual  
1125 message MAY be mutually authenticated and integrity protected, depending on the environment of use.

1126 If the actual SAML protocol message is intended for a specific recipient, then the artifact's issuer MUST  
1127 authenticate the sender of the subsequent `<samlp:ArtifactResolve>` message before returning the  
1128 actual message.

1129 The transmission of an artifact to and from the user agent SHOULD be protected with confidentiality; SSL  
1130 3.0 or TLS 1.0 SHOULD be used. The callback request/response exchange that returns the actual  
1131 message MAY be protected, depending on the environment of use.

1132 In general, this binding relies on the artifact as a hard-to-forge short-term reference and applies other  
1133 security measures to the callback request/response that returns the actual message. All artifacts MUST  
1134 have a single-use semantic enforced by the artifact issuer. Furthermore, it is RECOMMENDED that  
1135 artifact receivers also enforce a single-use semantic on the artifact values they receive, to prevent an  
1136 attacker from interfering with the resolution of an artifact by a user agent and then resubmitting it to the  
1137 artifact receiver.

1138 Note also that there is no mechanism defined to protect the integrity of the relationship between the  
1139 artifact and the "RelayState" value, if any. That is, an attacker can potentially recombine a pair of valid  
1140 HTTP responses by switching the "RelayState" values associated with each artifact. As a result, the  
1141 producer/consumer of "RelayState" information MUST take care not to associate sensitive state  
1142 information with the "RelayState" value without taking additional precautions (such as based on the  
1143 information in the SAML protocol message retrieved via artifact).

### 1144 **3.6.6 Error Reporting**

1145 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD  
1146 return a response message with a second-level `<samlp:StatusCode>` value of  
1147 `urn:oasis:names:tc:SAML:2.0:status:RequestDenied`.

1148 HTTP interactions during the message exchange MUST NOT use HTTP error status codes to indicate  
1149 failures in SAML processing, since the user agent is not a full party to the SAML protocol exchange.

1150 If the issuer of an artifact receives a `<samlp:ArtifactResolve>` message that it can understand, it  
1151 MUST return a `<samlp:ArtifactResponse>` with a `<samlp:StatusCode>` value of  
1152 `urn:oasis:names:tc:SAML:2.0:status:Success`, even if it does not return the corresponding  
1153 message (for example because the artifact requester is not authorized to receive the message or the  
1154 artifact is no longer valid).

1155 For more information about SAML status codes, see the SAML assertions and protocols specification  
1156 [SAMLCore].

### 1157 **3.6.7 Metadata Considerations**

1158 Support for the HTTP Artifact binding SHOULD be reflected by indicating URL endpoints at which  
1159 requests and responses for a particular protocol or profile should be sent. Either a single endpoint or  
1160 distinct request and response endpoints MAY be supplied. One or more indexed endpoints for processing  
1161 `<samlp:ArtifactResolve>` messages SHOULD also be described.

### 1162 **3.6.8 Example SAML Message Exchange Using HTTP Artifact**

1163 In this example, a `<LogoutRequest>` and `<LogoutResponse>` message pair are exchanged using the  
1164 HTTP Artifact binding, with the artifact resolution taking place using the SOAP binding bound to HTTP.

1165 First, here are the actual SAML protocol messages being exchanged:

```
1166 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
1167 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
1168 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
1169 21T19:00:49Z" Version="2.0">  
1170 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
1171 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
1172 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
1173 <samlp:SessionIndex>1</samlp:SessionIndex>  
1174 </samlp:LogoutRequest>
```

```
1175 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
1176 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
1177 ID="b0730d21b628110d8b7e004005b13a2b"  
1178 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"
```

```

1179     IssueInstant="2004-01-21T19:00:49Z" Version="2.0">
1180     <Issuer>https://ServiceProvider.com/SAML</Issuer>
1181     <samlp:Status>
1182         <samlp:StatusCode
1183 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1184     </samlp:Status>
1185 </samlp:LogoutResponse>

```

1186 The initial HTTP request from the user agent in step 1 is not defined by this binding. To initiate the logout  
1187 protocol exchange, the SAML requester returns the following HTTP response, containing a SAML artifact.  
1188 Note that the line feeds in the HTTP Location header below are a result of document formatting, and  
1189 there are no line feeds in the actual header value.

```

1190 HTTP/1.1 302 Object Moved
1191 Date: 21 Jan 2004 07:00:49 GMT
1192 Location:
1193 https://ServiceProvider.com/SAML/SLO/Browser?SAMLart=AAQAADWNEw5VT47wcO4z
1194 X%2FiEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU%
1195 3D&RelayState=0043bfc1bc45110dae17004005b13a2b
1196 Content-Type: text/html; charset=iso-8859-1

```

1197 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact  
1198 Resolution protocol and the SOAP binding in steps 3 and 4, as follows:

1199 Step 3:

```

1200 POST /SAML/Artifact/Resolve HTTP/1.1
1201 Host: IdentityProvider.com
1202 Content-Type: text/xml
1203 Content-Length: nnn
1204 SOAPAction: http://www.oasis-open.org/committees/security
1205 <SOAP-ENV:Envelope
1206     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1207     <SOAP-ENV:Body>
1208         <samlp:ArtifactResolve
1209             xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1210             xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1211             ID="_6c3a4f8b9c2d" Version="2.0"
1212             IssueInstant="2004-01-21T19:00:49Z">
1213             <Issuer>https://ServiceProvider.com/SAML</Issuer>
1214             <Artifact>
1215                 AAQAADWNEw5VT47wcO4zX/iEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU=
1216             </Artifact>
1217         </samlp:ArtifactResolve>
1218     </SOAP-ENV:Body>
1219 </SOAP-ENV:Envelope>

```

1220 Step 4:

```

1221 HTTP/1.1 200 OK
1222 Date: 21 Jan 2004 07:00:49 GMT
1223 Content-Type: text/xml
1224 Content-Length: nnnn
1225 <SOAP-ENV:Envelope
1226     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1227     <SOAP-ENV:Body>
1228         <samlp:ArtifactResponse
1229             xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1230             xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1231             ID="_FQvGknDfws2Z" Version="2.0"
1232             InResponseTo="_6c3a4f8b9c2d"
1233             IssueInstant="2004-01-21T19:00:49Z">
1234             <Issuer>https://IdentityProvider.com/SAML</Issuer>
1235             <samlp:Status>
1236                 <samlp:StatusCode
1237 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1238             </samlp:Status>

```

```

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

```

1250 After any unspecified interactions may have taken place, the SAML responder returns a second SAML  
1251 artifact in its HTTP response in step 6:

```

1252 HTTP/1.1 302 Object Moved
1253 Date: 21 Jan 2004 07:05:49 GMT
1254 Location:
1255 https://IdentityProvider.com/SAML/SLO/Response?SAMLart=AAQAAFQIZXv5%
1256 2BQaBaE5qYurHWJ01nAgLAsqfnyidHIggbFU0mlSGFTyQiPc%
1257 3D&RelayState=0043bfc1bc45110dae17004005b13a2b
1258 Content-Type: text/html; charset=iso-8859-1

```

1259 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact  
1260 Resolution protocol and the SOAP binding in steps 7 and 8, as follows:

1261 Step 7:

```

1262 POST /SAML/Artifact/Resolve HTTP/1.1
1263 Host: ServiceProvider.com
1264 Content-Type: text/xml
1265 Content-Length: nnn
1266 SOAPAction: http://www.oasis-open.org/committees/security
1267 <SOAP-ENV:Envelope
1268     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1269     <SOAP-ENV:Body>
1270         <samlp:ArtifactResolve
1271             xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1272             xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1273             ID="_ec36fa7c39" Version="2.0"
1274             IssueInstant="2004-01-21T19:05:49Z">
1275             <Issuer>https://IdentityProvider.com/SAML</Issuer>
1276             <Artifact>
1277                 AAQAAFQIZXv5+QaBaE5qYurHWJ01nAgLAsqfnyidHIggbFU0mlSGFTyQiPc=
1278             </Artifact>
1279         </samlp:ArtifactResolve>
1280     </SOAP-ENV:Body>
1281 </SOAP-ENV:Envelope>

```

1282 Step 8:

```

1283 HTTP/1.1 200 OK
1284 Date: 21 Jan 2004 07:05:49 GMT
1285 Content-Type: text/xml
1286 Content-Length: nnnn
1287 <SOAP-ENV:Envelope
1288     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1289     <SOAP-ENV:Body>
1290         <samlp:ArtifactResponse
1291             xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1292             xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1293             ID="_FQvGknDfws2Z" Version="2.0"
1294             InResponseTo="_ec36fa7c39"
1295             IssueInstant="2004-01-21T19:05:49Z">
1296             <Issuer>https://ServiceProvider.com/SAML</Issuer>
1297             <samlp:Status>
1298                 <samlp:StatusCode

```

```

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

```

## 1314 3.7 SAML URI Binding

1315 URIs are a protocol-independent means of referring to a resource. This binding is not a general SAML  
1316 request/response binding, but rather supports the encapsulation of a `<samlp:AssertionIDRequest>`  
1317 message with a single `<saml:AssertionIDRef>` into the resolution of a URI. The result of a successful  
1318 request is a SAML `<saml:Assertion>` element (but not a complete SAML response).

1319 Like SOAP, URI resolution can occur over multiple underlying transports. This binding has transport-  
1320 independent aspects, but also calls out the use of HTTP with SSL 3.0 or TLS 1.0 as REQUIRED  
1321 (mandatory to implement).

### 1322 3.7.1 Required Information

1323 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:URI

1324 **Contact information:** [security-services-comment@lists.oasis-open.org](mailto:security-services-comment@lists.oasis-open.org)

1325 **Description:** Given below.

1326 **Updates:** None

### 1327 3.7.2 Protocol-Independent Aspects of the SAML URI Binding

1328 The following sections define aspects of the SAML URI binding that are independent of the underlying  
1329 transport protocol of the URI resolution process.

#### 1330 3.7.2.1 Basic Operation

1331 A SAML URI reference identifies a specific SAML assertion. The result of resolving the URI MUST be a  
1332 message containing the assertion, or a transport-specific error. The specific format of the message  
1333 depends on the underlying transport protocol. If the transport protocol permits the returned content to be  
1334 described, such as HTTP 1.1 [RFC2616], then the assertion MAY be encoded in whatever format is  
1335 permitted. If not, the assertion MUST be returned in a form which can be unambiguously interpreted as or  
1336 transformed into an XML serialization of the assertion.

1337 It MUST be the case that if the same URI reference is resolved in the future, then either the same SAML  
1338 assertion, or an error, is returned. That is, the reference MAY be persistent but MUST consistently  
1339 reference the same assertion, if any.

#### 1340 3.7.3 Security Considerations

1341 Indirect use of a SAML assertion presents dangers if the binding of the reference to the result is not  
1342 secure. The particular threats and their severity depend on the use to which the assertion is being put. In

1343 general, the result of resolving a URI reference to a SAML assertion SHOULD only be trusted if the  
1344 requester can be certain of the identity of the responder and that the contents have not been modified in  
1345 transit.

1346 It is often not sufficient that the assertion itself be signed, because URI references are by their nature  
1347 somewhat opaque to the requester. The requester SHOULD have independent means to insure that the  
1348 assertion returned is actually the one that is represented by the URI; this is accomplished by both  
1349 authenticating the responder and relying on the integrity of the response.

### 1350 **3.7.4 MIME Encapsulation**

1351 For resolution protocols that support MIME as a content description and packaging mechanism, the  
1352 resulting assertion SHOULD be returned as a MIME entity of type `application/samlassertion+xml`,  
1353 as defined by [\[SAMLmime\]](#).

### 1354 **3.7.5 Use of HTTP URIs**

1355 A SAML authority that claims conformance to the SAML URI binding MUST implement support for HTTP.  
1356 This section describes certain specifics of using HTTP URIs, including URI syntax, HTTP headers, and  
1357 error reporting.

#### 1358 **3.7.5.1 URI Syntax**

1359 In general, there are no restrictions on the permissible syntax of a SAML URI reference as long as the  
1360 SAML authority responsible for the reference creates the message containing it. However, authorities  
1361 MUST support a URL endpoint at which an HTTP request can be sent with a single query string  
1362 parameter named `ID`. There MUST be no query string in the endpoint URL itself independent of this  
1363 parameter.

1364 For example, if the documented endpoint at an authority is "<https://saml.example.edu/assertions>", a  
1365 request for an assertion with an `ID` of `abcde` can be sent to:

1366 `https://saml.example.edu/assertions?ID=abcde`

1367 Note that the use of wildcards is not allowed for such ID queries.

#### 1368 **3.7.5.2 HTTP and Caching Considerations**

1369 HTTP proxies MUST NOT cache SAML assertions. To insure this, the following rules SHOULD be  
1370 followed.

1371 When returning SAML assertions using HTTP 1.1, HTTP responders SHOULD:

- 1372 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 1373 • Include a `Pragma` header field set to "no-cache".

#### 1374 **3.7.5.3 Security Considerations**

1375 [RFC2617] describes possible attacks in the HTTP environment when basic or message-digest  
1376 authentication schemes are used.

1377 Use of SSL 3.0 or TLS 1.0 is STRONGLY RECOMMENDED as a means of authentication, integrity  
1378 protection, and confidentiality.

#### 1379 **3.7.5.4 Error Reporting**

1380 As an HTTP protocol exchange, the appropriate HTTP status code SHOULD be used to indicate the result

1381 of a request. For example, a SAML responder that refuses to perform a message exchange with the  
1382 SAML requester SHOULD return a "403 Forbidden" response. If the assertion specified is unknown to  
1383 the responder, then a "404 Not Found" response SHOULD be returned. In these cases, the content of  
1384 the HTTP body is not significant.

### 1385 **3.7.5.5 Metadata Considerations**

1386 Support for the URI binding over HTTP SHOULD be reflected by indicating a URL endpoint at which  
1387 requests for arbitrary assertions are to be sent.

### 1388 **3.7.5.6 Example SAML Message Exchange Using an HTTP URI**

1389 Following is an example of a request for an assertion.

```
1390 GET /SamlService?ID=abcde HTTP/1.1  
1391 Host: www.example.com
```

1392 Following is an example of the corresponding response, which supplies the requested assertion.

```
1393 HTTP/1.1 200 OK  
1394 Content-Type: application/samlassertion+xml  
1395 Cache-Control: no-cache, no-store  
1396 Pragma: no-cache  
1397 Content-Length: nnnn  
  
1398 <saml:Assertion ID="abcde" ...>  
1399 ...  
1400 </saml:Assertion>
```

---

## 4 References

1401

- 1402     **[AES]**             FIPS-197, Advanced Encryption Standard (AES), available from  
1403                     <http://www.nist.gov/>.
- 1404     **[Anders]**           A suggestion on how to implement SAML browser bindings without using  
1405                     “Artifacts”, <http://www.x-obi.com/OBI400/andersr-browser-artifact.ppt>.
- 1406     **[CoreAssnEx]**       Core Assertions Architecture, Examples and Explanations, [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-phill-07.pdf)  
1407                     [open.org/committees/security/docs/draft-sstc-core-phill-07.pdf](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-phill-07.pdf).
- 1408     **[HTML401]**         HTML 4.01 Specification, W3C Recommendation 24 December 1999,  
1409                     <http://www.w3.org/TR/html4>.
- 1410     **[XHTML]**           XHTML 1.0 The Extensible HyperText Markup Language (Second Edition),  
1411                     <http://www.w3.org/TR/xhtml1/>.
- 1412     **[Liberty]**          The Liberty Alliance Project, <http://www.projectliberty.org>.
- 1413     **[MSURL]**           Microsoft technical support article,  
1414                     <http://support.microsoft.com/support/kb/articles/Q208/4/27.ASP>.
- 1415     **[PAOS]**            Aarts, R., “Liberty Reverse HTTP Binding for SOAP Specification”, Version: 1.0,  
1416                     <https://www.projectliberty.org/specs/liberty-paos-v1.0.pdf>
- 1417     **[Rescorla-Sec]**    E. Rescorla et al., *Guidelines for Writing RFC Text on Security Considerations*,  
1418                     <http://www.ietf.org/internet-drafts/draft-iab-sec-cons-03.txt>.
- 1419     **[RFC1952]**         GZIP file format specification version 4.3, <http://www.ietf.org/rfc/rfc1952.txt>
- 1420     **[RFC1738]**         Uniform Resource Locators (URL), <http://www.ietf.org/rfc/rfc1738.txt>
- 1421     **[RFC1750]**         Randomness Recommendations for Security. <http://www.ietf.org/rfc/rfc1750.txt>
- 1422     **[RFC1945]**         Hypertext Transfer Protocol -- HTTP/1.0, <http://www.ietf.org/rfc/rfc1945.txt>.
- 1423     **[RFC2045]**         Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet  
1424                     Message Bodies, <http://www.ietf.org/rfc/rfc2045.txt>
- 1425     **[RFC2119]**         S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF  
1426                     RFC 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>.
- 1427     **[RFC2246]**         The TLS Protocol Version 1.0, <http://www.ietf.org/rfc/rfc2246.txt>.
- 1428     **[RFC2279]**         UTF-8, a transformation format of ISO 10646, <http://www.ietf.org/rfc/rfc2279.txt>.
- 1429     **[RFC2616]**         Hypertext Transfer Protocol -- HTTP/1.1, <http://www.ietf.org/rfc/rfc2616.txt>.
- 1430     **[RFC2617]**         *HTTP Authentication: Basic and Digest Access Authentication*, IETF RFC 2617,  
1431                     <http://www.ietf.org/rfc/rfc2617.txt>.
- 1432     **[SAMLCore]**         S. Cantor et al., *Assertions and Protocols for the OASIS Security Assertion*  
1433                     *Markup Language (SAML) V2.0*. OASIS SSTC, August 2004. Document ID sstc-  
1434                     saml-core-2.0-cd-01. See <http://www.oasis-open.org/committees/security/>.
- 1435     **[SAMLGloss]**        J. Hodges et al., *Glossary for the OASIS Security Assertion Markup Language*  
1436                     *(SAML) V2.0*. OASIS SSTC, August 2004. Document ID sstc-saml-glossary-2.0-  
1437                     cd-01. See <http://www.oasis-open.org/committees/security/>.
- 1438     **[SAMLProfile]**      S. Cantor et al., *Profiles for the OASIS Security Assertion Markup Language*  
1439                     *(SAML) V2.0*. OASIS SSTC, August 2004. Document ID sstc-saml-profiles-2.0-  
1440                     cd-01. See <http://www.oasis-open.org/committees/security/>.
- 1441     **[SAMLMeta]**        S. Cantor et al., *Metadata for the OASIS Security Assertion Markup Language*  
1442                     *(SAML) V2.0*. OASIS SSTC, August 2004. Document ID sstc-saml-metadata-2.0-  
1443                     cd-01. See <http://www.oasis-open.org/committees/security/>.
- 1444     **[SAMLmime]**         <http://www.ietf.org/internet-drafts/draft-hodges-saml-mediatype-01.txt>.

1445	<b>[SAMLSecure]</b>	F. Hirsch et al., <i>Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0</i> . OASIS SSTC, August 2004.
1446		Document ID sstc-saml-sec-consider-2.0-cd-01. See <a href="http://www.oasis-open.org/committees/security/">http://www.oasis-open.org/committees/security/</a> .
1447		
1448		
1449	<b>[SAMLReqs]</b>	Darren Platt et al., <i>SAML Requirements and Use Cases</i> , OASIS, April 2002, <a href="http://www.oasis-open.org/committees/security/">http://www.oasis-open.org/committees/security/</a> .
1450		
1451	<b>[SAMLWeb]</b>	OASIS Security Services Technical Committee website, <a href="http://www.oasis-open.org/committees/security/">http://www.oasis-open.org/committees/security/</a> .
1452		
1453	<b>[SESSION]</b>	RL "Bob" Morgan, Support of target web server sessions in Shibboleth, <a href="http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-session-00.txt">http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-session-00.txt</a>
1454		
1455		
1456	<b>[ShibMarlena]</b>	Marlena Erdos, Shibboleth Architecture DRAFT v1.1, <a href="http://shibboleth.internet2.edu/draft-internet2-shibboleth-arch-v05.html">http://shibboleth.internet2.edu/draft-internet2-shibboleth-arch-v05.html</a> .
1457		
1458	<b>[SOAP1.1]</b>	D. Box et al., <i>Simple Object Access Protocol (SOAP) 1.1</i> , World Wide Web Consortium Note, May 2000, <a href="http://www.w3.org/TR/SOAP">http://www.w3.org/TR/SOAP</a> .
1459		
1460	<b>[SOAP-PRIMER]</b>	N. Mitra, SOAP Version 1.2 Part 0: Primer, W3C Recommendation 24 June 2003, <a href="http://www.w3.org/TR/soap12-part0/">http://www.w3.org/TR/soap12-part0/</a>
1461		
1462	<b>[SSL3]</b>	A. Frier et al., <i>The SSL 3.0 Protocol</i> , Netscape Communications Corp, November 1996.
1463		
1464	<b>[WEBSSO]</b>	RL "Bob" Morgan, Interactions between Shibboleth and local-site web sign-on services, <a href="http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-websso-00.txt">http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-websso-00.txt</a>
1465		
1466		
1467	<b>[WSS-SAML]</b>	P. Hallam-Baker et al., <i>Web Services Security: SAML Token Profile</i> , OASIS, March 2003, <a href="http://www.oasis-open.org/committees/wss">http://www.oasis-open.org/committees/wss</a> .
1468		
1469	<b>[WSS-Sec]</b>	A. Nadalin et al., <i>Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)</i> , OASIS Standard 200401, March 2004, <a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf</a>
1470		
1471		
1472	<b>[XMLSig]</b>	D. Eastlake et al., <i>XML-Signature Syntax and Processing</i> , World Wide Web Consortium, <a href="http://www.w3.org/TR/xmlsig-core/">http://www.w3.org/TR/xmlsig-core/</a> .
1473		

---

## 1474 **Appendix A. Acknowledgments**

1475 The editors would like to acknowledge the contributions of the OASIS Security Services Technical  
1476 Committee, whose voting members at the time of publication were:

- 1477 • Conor Cahill, AOL
- 1478 • Hal Lockhart, BEA Systems
- 1479 • Rick Randall, Booz Allen Hamilton
- 1480 • Ronald Jacobson, Computer Associates
- 1481 • Gavenraj Sodhi, Computer Associates
- 1482 • Tim Alsop, CyberSafe Limited
- 1483 • Paul Madsen, Entrust
- 1484 • Carolina Canales-Valenzuela, Ericsson
- 1485 • Dana Kaufman, Forum Systems
- 1486 • Irving Reid, Hewlett-Packard
- 1487 • Paula Austel, IBM
- 1488 • Maryann Hondo, IBM
- 1489 • Michael McIntosh, IBM
- 1490 • Anthony Nadalin, IBM
- 1491 • Nick Ragouzis, Individual
- 1492 • Scott Cantor, Internet2
- 1493 • Bob Morgan, Internet2
- 1494 • Prateek Mishra, Netegrity
- 1495 • Forest Yin, Netegrity
- 1496 • Peter Davis, Neustar
- 1497 • Frederick Hirsch, Nokia
- 1498 • John Kemp, Nokia
- 1499 • Senthil Sengodan, Nokia
- 1500 • Scott Kiestler, Novell
- 1501 • Steve Anderson, OpenNetwork
- 1502 • Ari Kermaier, Oracle
- 1503 • Vamsi Motukuru, Oracle
- 1504 • Darren Platt, Ping Identity
- 1505 • Jim Lien, RSA Security
- 1506 • John Linn, RSA Security
- 1507 • Rob Philpott, RSA Security
- 1508 • Dipak Chopra, SAP
- 1509 • Jahan Moreh, Sigaba
- 1510 • Bhavna Bhatnagar, Sun Microsystems
- 1511 • Jeff Hodges, Sun Microsystems
- 1512 • Eve Maler, Sun Microsystems
- 1513 • Ronald Monzillo, Sun Microsystems
- 1514 • Emily Xu, Sun Microsystems
- 1515 • Mike Beach, Boeing

- 1516 • Greg Whitehead, Trustgenix
- 1517 • James Vanderbeek, Vodafone

1518 The editors also would like to acknowledge the following people for their contributions to previous versions  
1519 of the OASIS Security Assertions Markup Language Standard:

- 1520 • Stephen Farrell, Baltimore Technologies
- 1521 • David Orchard, BEA Systems
- 1522 • Krishna Sankar, Cisco Systems
- 1523 • Zahid Ahmed, CommerceOne
- 1524 • Carlisle Adams, Entrust
- 1525 • Tim Moses, Entrust
- 1526 • Nigel Edwards, Hewlett-Packard
- 1527 • Joe Pato, Hewlett-Packard
- 1528 • Bob Blakley, IBM
- 1529 • Marlena Erdos, IBM
- 1530 • Marc Chanliau, Netegrity
- 1531 • Chris McLaren, Netegrity
- 1532 • Lynne Rosenthal, NIST
- 1533 • Mark Skall, NIST
- 1534 • Simon Godik, Overxeer
- 1535 • Charles Norwood, SAIC
- 1536 • Evan Prodromou, Securant
- 1537 • Robert Griffin, RSA Security (former editor)
- 1538 • Sai Allarvarpu, Sun Microsystems
- 1539 • Chris Ferris, Sun Microsystems
- 1540 • Emily Xu, Sun Microsystems
- 1541 • Mike Myers, Traceroute Security
- 1542 • Phillip Hallam-Baker, VeriSign (former editor)
- 1543 • James Vanderbeek, Vodafone
- 1544 • Mark O'Neill, Vordel
- 1545 • Tony Palmer, Vordel

1546 Finally, the editors wish to acknowledge the following people for their contributions of material used as  
1547 input to the OASIS Security Assertions Markup Language specifications:

- 1548 • Thomas Gross, IBM
- 1549 • Birgit Pfitzmann, IBM

---

## 1550 Appendix B. Notices

1551 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that  
1552 might be claimed to pertain to the implementation or use of the technology described in this document or  
1553 the extent to which any license under such rights might or might not be available; neither does it represent  
1554 that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to  
1555 rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made  
1556 available for publication and any assurances of licenses to be made available, or the result of an attempt  
1557 made to obtain a general license or permission for the use of such proprietary rights by implementors or  
1558 users of this specification, can be obtained from the OASIS Executive Director.

1559 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or  
1560 other proprietary rights which may cover technology that may be required to implement this specification.  
1561 Please address the information to the OASIS Executive Director.

1562 **Copyright © OASIS Open 2004. All Rights Reserved.**

1563 This document and translations of it may be copied and furnished to others, and derivative works that  
1564 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and  
1565 distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and  
1566 this paragraph are included on all such copies and derivative works. However, this document itself may  
1567 not be modified in any way, such as by removing the copyright notice or references to OASIS, except as  
1568 needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights  
1569 defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it  
1570 into languages other than English.

1571 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors  
1572 or assigns.

1573 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
1574 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY  
1575 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR  
1576 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.