

---

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

Working Draft 05, 12 FebruaryDecember 20097

## Document identifier:

sstc-saml-bindings-errata-2.0-wd-054

## 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 (errata editor)

## Contributors to the Errata:

Rob Philpott, EMC Corporation  
Nick Ragouzis, Enosis Group  
Thomas Wisniewski, Entrust  
Greg Whitehead, HP  
Heather Hinton, IBM  
Connor P. Cahill, Intel  
Scott Cantor, Internet2  
Nate Klingenstein, Internet2  
RL 'Bob' Morgan, Internet2  
John Bradley, Individual  
Jeff Hodges, Individual  
Joni Brennan, Liberty Alliance  
Eric Tiffany, Liberty Alliance  
Thomas Hardjono, M.I.T.  
Tom Scavo, NCSA  
Peter Davis, NeuStar, Inc.  
Frederick Hirsch, Nokia Corporation  
Paul Madsen, NTT Corporation  
Ari Kermaier, Oracle Corporation  
Hal Lockhart, Oracle Corporation  
Prateek Mishra, Oracle Corporation  
Brian Campbell, Ping Identity  
Anil Saldhana, Red Hat Inc.  
Jim Lien, RSA Security  
Jahan Moreh, Sigaba  
Kent Spaulding, Skyworth TTG Holdings Limited  
Emily Xu, Sun Microsystems  
David Staggs, Veteran's Health Administration

## SAML V2.0 Contributors:

Conor P. Cahill, AOL  
John Hughes, Atos Origin

48 Hal Lockhart, BEA Systems  
49 Michael Beach, Boeing  
50 Rebekah Metz, Booz Allen Hamilton  
51 Rick Randall, Booz Allen Hamilton  
52 Thomas Wisniewski, Entrust  
53 Irving Reid, Hewlett-Packard  
54 Paula Austel, IBM  
55 Maryann Hondo, IBM  
56 Michael McIntosh, IBM  
57 Tony Nadalin, IBM  
58 Nick Ragouzis, Individual  
59 Scott Cantor, Internet2  
60 RL 'Bob' Morgan, Internet2  
61 Peter C Davis, Neustar  
62 Jeff Hodges, Neustar  
63 Frederick Hirsch, Nokia  
64 John Kemp, Nokia  
65 Paul Madsen, NTT  
66 Steve Anderson, OpenNetwork  
67 Prateek Mishra, Principal Identity  
68 John Linn, RSA Security  
69 Rob Philpott, RSA Security  
70 Jahan Moreh, Sigaba  
71 Anne Anderson, Sun Microsystems  
72 Eve Maler, Sun Microsystems  
73 Ron Monzillo, Sun Microsystems  
74 Greg Whitehead, Trustgenix

75 **Abstract:**

76 The SAML V2.0 Bindings specification defines protocol bindings for the use of SAML assertions  
77 and request-response messages in communications protocols and frameworks. This document,  
78 known as an “errata composite”, combines corrections to reported errata with the original  
79 specification text. By design, the corrections are limited to clarifications of ambiguous or  
80 conflicting specification text. This document shows deletions from the original specification as  
81 struck-through text, and additions as colored underlined text. The “[*Err*]” designations embedded  
82 in the text refer to particular errata and their dispositions.

83 **Status:**

84 This errata composite document is a **working draft** based on the [original](#) OASIS Standard  
85 document that had been produced by the Security Services Technical Committee and approved  
86 by the OASIS membership on 1 March 2005. While the errata corrections appearing here are  
87 non-normative, they reflect changes specified by the Approved Errata document (currently at  
88 Working Draft revision 02), which is on an OASIS standardization track. In case of any  
89 discrepancy between this document and the Approved Errata, the latter has precedence. ~~See also~~  
90 ~~the Errata Working Document (currently at revision 39), which provides background on the~~  
91 ~~changes specified here.~~

92 This document includes corrections for errata E1, E2, E4, E19, E21, E24, E31, E57, ~~and E59, and~~  
93 ~~E74.~~

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

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

---

# Table of Contents

101

102	1 Introduction.....	5
103	1.1 Protocol Binding Concepts.....	5
104	1.2 Notation.....	5
105	2 Guidelines for Specifying Additional Protocol Bindings.....	7
106	3 Protocol Bindings.....	8
107	3.1 General Considerations.....	8
108	3.1.1 Use of RelayState.....	8
109	3.1.2 Security.....	8
110	3.1.2.1 Use of SSL 3.0 or TLS 1.0.....	8
111	3.1.2.2 Data Origin Authentication.....	8
112	3.1.2.3 Message Integrity.....	8
113	3.1.2.4 Message Confidentiality.....	9
114	3.1.2.5 Security Considerations.....	9
115	3.2 SAML SOAP Binding.....	9
116	3.2.1 Required Information.....	9
117	3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding.....	10
118	3.2.2.1 Basic Operation.....	10
119	3.2.2.2 SOAP Headers.....	10
120	3.2.3 Use of SOAP over HTTP.....	11
121	3.2.3.1 HTTP Headers.....	11
122	3.2.3.2 Caching.....	11
123	3.2.3.3 Error Reporting.....	11
124	3.2.3.4 Metadata Considerations.....	12
125	3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP.....	12
126	3.3 Reverse SOAP (PAOS) Binding.....	13
127	3.3.1 Required Information.....	13
128	3.3.2 Overview.....	13
129	3.3.3 Message Exchange.....	13
130	3.3.3.1 HTTP Request, SAML Request in SOAP Response.....	14
131	3.3.3.2 SAML Response in SOAP Request, HTTP Response.....	15
132	3.3.4 Caching.....	15
133	3.3.5 Security Considerations.....	15
134	3.3.5.1 Error Reporting.....	15
135	3.3.5.2 Metadata Considerations.....	15
136	3.4 HTTP Redirect Binding.....	15
137	3.4.1 Required Information.....	16
138	3.4.2 Overview.....	16
139	3.4.3 RelayState.....	16
140	3.4.4 Message Encoding.....	16
141	3.4.4.1 DEFLATE Encoding.....	17
142	3.4.5 Message Exchange.....	18
143	3.4.5.1 HTTP and Caching Considerations.....	19
144	3.4.5.2 Security Considerations.....	19
145	3.4.6 Error Reporting.....	20
146	3.4.7 Metadata Considerations.....	20
147	3.4.8 Example SAML Message Exchange Using HTTP Redirect.....	20

148	3.5 HTTP POST Binding.....	21
149	3.5.1 Required Information.....	21
150	3.5.2 Overview.....	21
151	3.5.3 RelayState.....	22
152	3.5.4 Message Encoding.....	22
153	3.5.5 Message Exchange.....	22
154	3.5.5.1 HTTP and Caching Considerations.....	23
155	3.5.5.2 Security Considerations.....	24
156	3.5.6 Error Reporting.....	24
157	3.5.7 Metadata Considerations.....	24
158	3.5.8 Example SAML Message Exchange Using HTTP POST.....	24
159	3.6 HTTP Artifact Binding.....	26
160	3.6.1 Required Information.....	27
161	3.6.2 Overview.....	27
162	3.6.3 Message Encoding.....	27
163	3.6.3.1 RelayState.....	27
164	3.6.3.2 URL Encoding.....	27
165	3.6.3.3 Form Encoding.....	28
166	3.6.4 Artifact Format.....	28
167	3.6.4.1 Required Information.....	29
168	3.6.4.2 Format Details.....	29
169	3.6.5 Message Exchange.....	29
170	3.6.5.1 HTTP and Caching Considerations.....	31
171	3.6.5.2 Security Considerations.....	31
172	3.6.6 Error Reporting.....	32
173	3.6.7 Metadata Considerations.....	32
174	3.6.8 Example SAML Message Exchange Using HTTP Artifact.....	32
175	3.7 SAML URI Binding.....	35
176	3.7.1 Required Information.....	35
177	3.7.2 Protocol-Independent Aspects of the SAML URI Binding.....	35
178	3.7.2.1 Basic Operation.....	35
179	3.7.3 Security Considerations.....	36
180	3.7.4 MIME Encapsulation.....	36
181	3.7.5 Use of HTTP URIs.....	36
182	3.7.5.1 URI Syntax.....	36
183	3.7.5.2 HTTP and Caching Considerations.....	36
184	3.7.5.3 Security Considerations.....	36
185	3.7.5.4 Error Reporting.....	37
186	3.7.5.5 Metadata Considerations.....	37
187	3.7.5.6 Example SAML Message Exchange Using an HTTP URI.....	37
188	4 References.....	38
189	Appendix A. Registration of MIME media type application/samlassertion+xml.....	40
190	Appendix B. Acknowledgments.....	44
191	Appendix C. Notices.....	46

---

# 1 Introduction

192

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

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

## 1.1 Protocol Binding Concepts

200

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

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

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

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

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

## 1.2 Notation

215

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

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

220 `Example code listings appear like this.`

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

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

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

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

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

---

## 2 Guidelines for Specifying Additional Protocol Bindings

227  
228

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

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

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

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

---

## 261 **3 Protocol Bindings**

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

### 263 **3.1 General Considerations**

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

#### 265 **3.1.1 Use of RelayState**

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

#### 273 **3.1.2 Security**

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

##### 276 **3.1.2.1 Use of SSL 3.0 or TLS 1.0**

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

##### 281 **3.1.2.2 Data Origin Authentication**

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

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

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

##### 291 **3.1.2.3 Message Integrity**

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

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



#### 297 **3.1.2.4 Message Confidentiality**

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

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

#### 303 **3.1.2.5 Security Considerations**

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

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

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

### 311 **3.2 SAML SOAP Binding**

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

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

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

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

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

#### 330 **3.2.1 Required Information**

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

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

333 **Description:** Given below.

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

## 335 3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding

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

### 339 3.2.2.1 Basic Operation

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

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

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

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

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

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

### 368 3.2.2.2 SOAP Headers

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

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

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

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

### 380 3.2.3 Use of SOAP over HTTP

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

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

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

#### 388 3.2.3.1 HTTP Headers

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

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

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

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

#### 400 3.2.3.2 Caching

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

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

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

406 When using HTTP 1.1, responders SHOULD:

- 407 • Include a `Cache-Control` header field set to "no-cache, no-store, must-revalidate,  
408 private".
- 409 • Include a `Pragma` header field set to "no-cache".
- 410 • NOT include a `Validator`, such as a `Last-Modified` or `ETag` header.

#### 411 3.2.3.3 Error Reporting

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

414 As described in [SOAP11] Section 6.2, in the case of a SOAP error while processing a SOAP request, the  
415 SOAP HTTP server MUST return a "500 Internal Server Error" response and include a SOAP  
416 message in the response with a SOAP `<SOAP-ENV: fault>` element. This type of error SHOULD be  
417 returned for SOAP-related errors detected before control is passed to the SAML processor, or when the  
418 SOAP processor reports an internal error (for example, the SOAP XML namespace is incorrect, the SAML  
419 schema cannot be located, the SAML processor throws an exception, and so on).

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

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

### 426 3.2.3.4 Metadata Considerations

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

### 430 3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP

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

```
433 POST /SamlService HTTP/1.1
434 Host: www.example.com
435 Content-Type: text/xml
436 Content-Length: nnn
437 SOAPAction: http://www.oasis-open.org/committees/security
438 <SOAP-ENV:Envelope
439   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
440   <SOAP-ENV:Body>
441     <samlp:AttributeQuery xmlns:samlp:="..."
442     xmlns:saml="..." xmlns:ds="..." ID="_6c3a4f8b9c2d" Version="2.0"
443     IssueInstant="2004-03-27T08:41:00Z"
444       <ds:Signature> ... </ds:Signature>
445       <saml:Subject>
446         ...
447       </saml:Subject>
448     </samlp:AttributeQuery>
449   </SOAP-ENV:Body>
450 </SOAP-ENV:Envelope>
```

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

```
453 HTTP/1.1 200 OK
454 Content-Type: text/xml
455 Content-Length: nnnn
456 <SOAP-ENV:Envelope
457   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
458   <SOAP-ENV:Body>
459     <samlp:Response xmlns:samlp:="..." xmlns:saml="..." xmlns:ds="..."
460     ID="_6c3a4f8b9c2d" Version="2.0" IssueInstant="2004-03-27T08:42:00Z">
461       <saml:Issuer>https://www.example.com/SAML</saml:Issuer>
462       <ds:Signature> ... </ds:Signature>
463       <Status>
464         <StatusCode Value="..." />
465       </Status>
466
467       <saml:Assertion>
468         <saml:Subject>
469           ...
470         </saml:Subject>
471         <saml:AttributeStatement>
472           ...
473         </saml:AttributeStatement>
474       </saml:Assertion>
475     </samlp:Response>
476   </SOAP-Env:Body>
```

## 478 3.3 Reverse SOAP (PAOS) Binding

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

### 482 3.3.1 Required Information

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

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

485 **Description:** Given below.

486 **Updates:** None.

### 487 3.3.2 Overview

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

### 495 3.3.3 Message Exchange

496 The PAOS binding includes two component message exchange patterns:

- 497 1. The HTTP requester sends an HTTP request to a SAML requester. The SAML requester responds  
498 with an HTTP response containing a SOAP envelope containing a SAML request message.
- 499 2. Subsequently, the HTTP requester sends an HTTP request to the original SAML requester  
500 containing a SOAP envelope containing a SAML response message. The SAML requester  
501 responds with an HTTP response, possibly in response to the original service request in step 1.

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

- 504 1. The client requests a service using an HTTP request.
- 505 2. The service provider responds with a SAML authentication request. This is sent using a SOAP  
506 request, carried in the HTTP response.
- 507 3. The client returns a SOAP response carrying a SAML authentication response. This is sent using a  
508 new HTTP request.
- 509 4. Assuming the service provider authentication and authorization is successful, the service provider  
510 may respond to the original service request in the HTTP response.

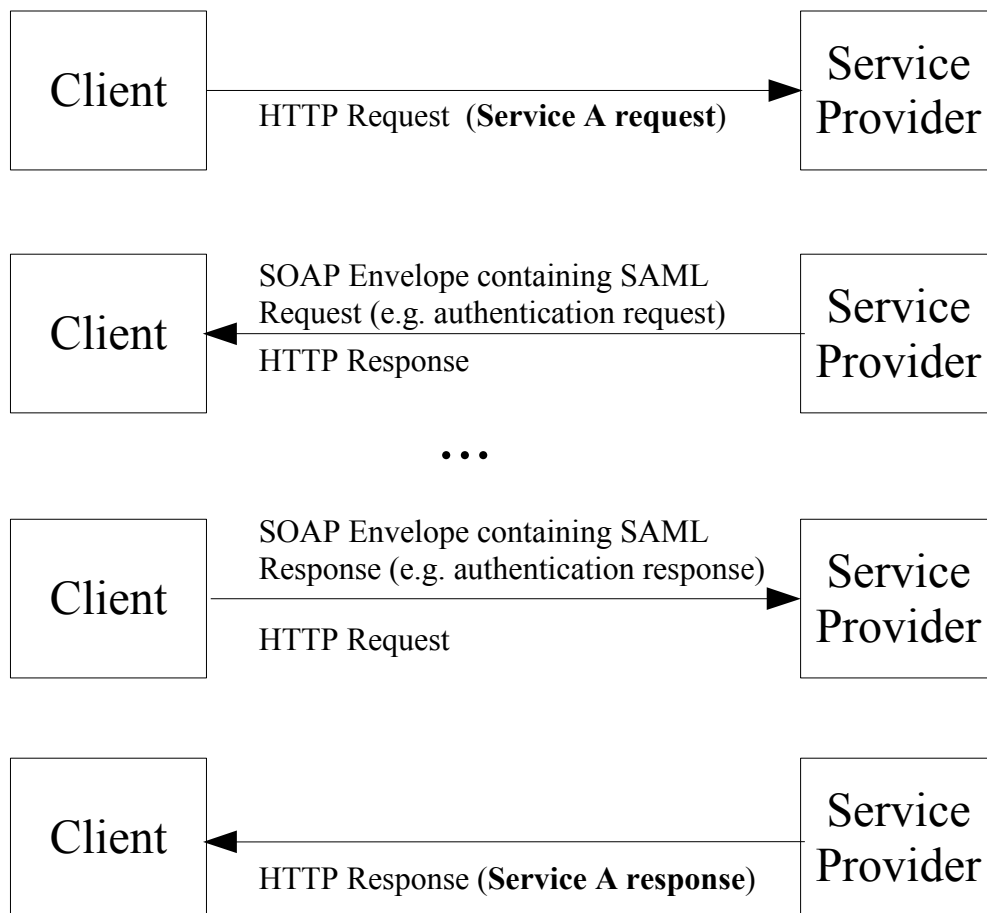


Figure 1: PAOS Binding Message Exchanges

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

- 513 • The HTTP `Accept` Header field MUST indicate an ability to accept the  
 514 “application/vnd.paos+xml” content type.
- 515 • The HTTP `PAOS` Header field MUST be present and specify the PAOS version with  
 516 “urn:liberty:paos:2003-08”[\[E21\]](#) at a minimum.

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

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

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

### 523 3.3.3.1 HTTP Request, SAML Request in SOAP Response

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

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



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

### 531 **3.3.3.2 SAML Response in SOAP Request, HTTP Response**

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

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

### 539 **3.3.4 Caching**

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

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

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

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

- 546 • Include a `Cache-Control` header field set to "no-cache, no-store, must-revalidate,  
547 private".
- 548 • Include a `Pragma` header field set to "no-cache".
- 549 • NOT include a `Validator`, such as a `Last-Modified` or `ETag` header.

### 550 **3.3.5 Security Considerations**

551 The HTTP requester in the PAOS binding may act as a SOAP intermediary and when it does, transport  
552 layer security for origin authentication, integrity and confidentiality may not meet end-end security  
553 requirements. In this case security at the SOAP message layer is [\[E31\]recommendedRECOMMENDED](#).

#### 554 **3.3.5.1 Error Reporting**

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

#### 558 **3.3.5.2 Metadata Considerations**

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

## 562 **3.4 HTTP Redirect Binding**

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

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

### 572 3.4.1 Required Information

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

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

575 **Description:** Given below.

576 **Updates:** None.

### 577 3.4.2 Overview

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

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

### 586 3.4.3 RelayState

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

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

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

### 601 3.4.4 Message Encoding

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



608 messages or content can or cannot be so encoded.

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

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

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

### 616 3.4.4.1 DEFLATE Encoding

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

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

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

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

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

```
647 SAMLRequest=value&RelayState=value&SigAlg=value  
648 SAMLResponse=value&RelayState=value&SigAlg=value
```

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

654 before being added.

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

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

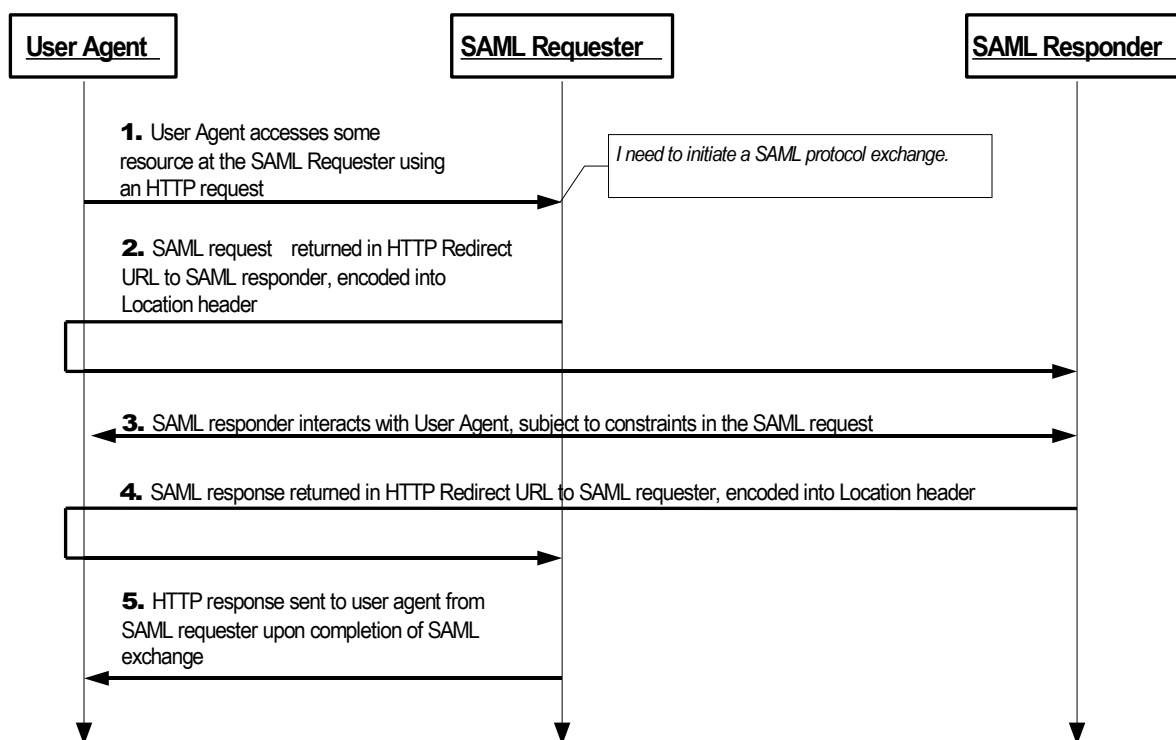
659 Note that when verifying signatures, the order of the query string parameters on the resulting URL to be  
660 verified is not prescribed by this binding. The parameters may appear in any order. Before verifying a  
661 signature, if any, the relying party MUST ensure that the parameter values to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

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

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

### 669 3.4.5 Message Exchange

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



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

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

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

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

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

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

### 693 **3.4.5.1 HTTP and Caching Considerations**

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

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

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

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

### 700 **3.4.5.2 Security Considerations**

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

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

708 This binding SHOULD NOT be used if the content of the request or response should not be exposed to  
709 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is  
710 OPTIONAL and depends on the environment of use. If confidentiality is necessary, SSL 3.0 [SSL3] or TLS  
711 1.0 [RFC2246] SHOULD be used to protect the message in transit between the user agent and the SAML  
712 requester and responder.

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

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

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

### 721 3.4.6 Error Reporting

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

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

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

### 729 3.4.7 Metadata Considerations

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

### 733 3.4.8 Example SAML Message Exchange Using HTTP Redirect

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

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

```
737 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
738 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
739 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
740 21T19:00:49Z" Version="2.0">  
741 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
742 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
743 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
744 <samlp:SessionIndex>1</samlp:SessionIndex>  
745 </samlp:LogoutRequest>
```

```
746 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
747 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
748 ID="b0730d21b628110d8b7e004005b13a2b"  
749 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"  
750 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">  
751 <Issuer>https://ServiceProvider.com/SAML</Issuer>  
752 <samlp:Status>  
753 <samlp:StatusCode  
754 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>  
755 </samlp:Status>  
756 </samlp:LogoutResponse>
```

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

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

```
765 Location: https://ServiceProvider.com/SAML/SLO/Browser?
766 SAMLRequest=fVfD58MwFH0f7D%2BUvGdNsq62oSsIQyhMEsc%2B
767 %2BJYlmRbWpObeyvz3puv2IMjyFM7HPedyK1DdsZdb%2F
768 %2BEHfLFfgwVMTt3RgTwzazIEJ72CFqRTnQWJWu7uH7dSLJjsg0ev%2FZFMlttiBWADtt6R
769 %2BSyJr9msiRH7070sCm31Mj%2Bo%2BC
770 %2B1KA5GLEWeZaogSQMw2MYBKodrIhjLkONU8FdeSsZkVr6T5M0GiHMjvWCknqZXZ2OoPxF7k
771 GnaGOuwxZ%2Fn4L9bY8NC
772 %2By4dulXpRXnxPcXizSZ58KfTeHujEWkNPZylsh9bAMYUj02Uiy3jCpTCMo5M1StVjmn9SO
773 150s191U6RV2Dp0vsLIy7NM7YU82r9B90PrvCf85W%2FwL8zSVQzAEAAA%3D
774 %3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F
775 %2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-
776 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE
777 Content-Type: text/html; charset=iso-8859-1
```

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

```
782 HTTP/1.1 302 Object Moved
783 Date: 21 Jan 2004 07:00:49 GMT
784 Location: https://IdentityProvider.com/SAML/SLO/Response?
785 SAMLResponse=fVfNa4QwEL0X%2Bh8k912TaDUGFUp7EbZQ6rKH3mKcbQVNJBOX
786 %2FvxaXQ9tYec0vHlv3nzkqIZ%2BlAf7YSf
787 %2FBjhagxB8Db1BuZQKMjkjrcIOpVEDoPRa1o8vB8n3VI70eqttT1bJbbJCB0c7a8j9XTBH9V
788 yQhqYRbTlrEi4Yo61oUqA0pvShYZHiDQkqs411tAVpeZPqSagNOKrOas4zzcW55Z1I4liJrTX
789 iBJVBr4wvCJ877ijbcXZkmaRUxtk7CU7gcB5mLu8pKVdvdvghd
790 %2Ben9iDIMA3CXTsOrs5euBbfXdgh%2F9snDK%2FEqW69Ye%2BUvGL%2F8CfbQnBS
791 %2FQS3z4QLW9aTloBIws0j%2FGoyAb9%2FV34Dw5k779IBAAA
792 %3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F
793 %2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-
794 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE
795 Content-Type: text/html; charset=iso-8859-1
```

## 796 3.5 HTTP POST Binding

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

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

### 802 3.5.1 Required Information

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

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

805 **Description:** Given below.

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

### 808 3.5.2 Overview

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

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

### 817 **3.5.3 RelayState**

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

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

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

### 832 **3.5.4 Message Encoding**

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

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

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

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

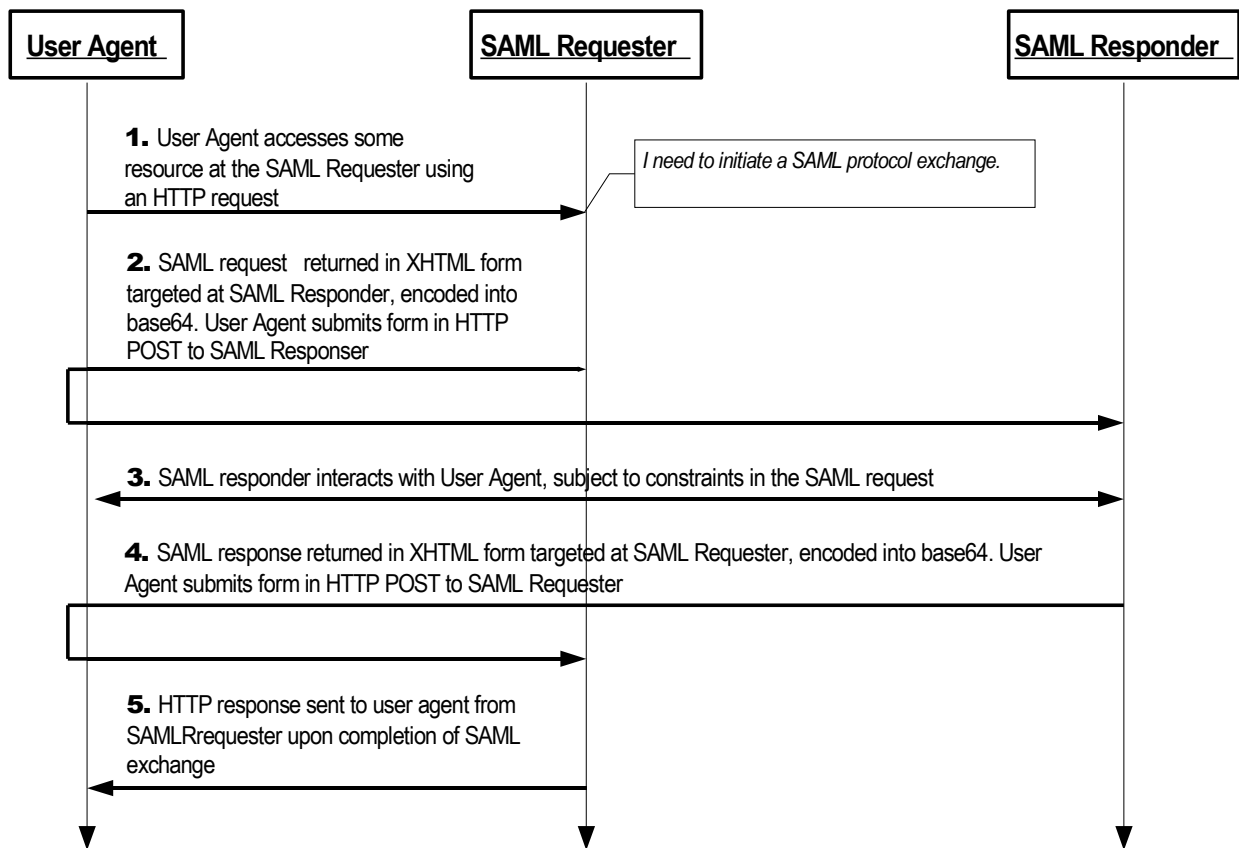
847 Any technique supported by the user agent MAY be used to cause the submission of the form, and any  
848 form content necessary to support this MAY be included, such as submit controls and client-side scripting  
849 commands. However, the recipient MUST be able to process the message without regard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

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

### 853 **3.5.5 Message Exchange**

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





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

### 875 3.5.5.1 HTTP and Caching Considerations

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

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

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

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

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

### 882 **3.5.5.2 Security Considerations**

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

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

891 This binding SHOULD NOT be used if the content of the request or response should not be exposed to  
892 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is  
893 OPTIONAL and depends on the environment of use. If confidentiality is necessary, SSL 3.0 [SSL3] or TLS  
894 1.0 [RFC2246] SHOULD be used to protect the message in transit between the user agent and the SAML  
895 requester and responder.

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

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

### 905 **3.5.6 Error Reporting**

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

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

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

### 913 **3.5.7 Metadata Considerations**

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

### 917 **3.5.8 Example SAML Message Exchange Using HTTP POST**

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

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







1035 **Contact information:** [security-services-comment@lists.oasis-open.org](mailto:security-services-comment@lists.oasis-open.org)

1036 **Description:** Given below.

1037 **Updates:** Effectively replaces the binding aspects of the Browser/Artifact profile in SAML V1.1  
1038 [SAML11Bind].

## 1039 **3.6.2 Overview**

1040 The HTTP Artifact binding is intended for cases in which the SAML requester and responder need to  
1041 communicate using an HTTP user agent as an intermediary, but the intermediary's limitations preclude or  
1042 discourage the transmission of an entire message (or message exchange) through it. This may be for  
1043 technical reasons or because of a reluctance to expose the message content to the intermediary (and if  
1044 the use of encryption is not practical).

1045 Note that because of the need to subsequently resolve the artifact using another synchronous binding,  
1046 such as SOAP, a direct communication path must exist between the SAML message sender and recipient  
1047 in the reverse direction of the artifact's transmission (the receiver of the message and artifact must be  
1048 able to send a `<samlp:ArtifactResolve>` request back to the artifact issuer). The artifact issuer must  
1049 also maintain state while the artifact is pending, which has implications for load-balanced environments.

## 1050 **3.6.3 Message Encoding**

1051 There are two methods of encoding an artifact for use with this binding. One is to encode the artifact into a  
1052 URL parameter and the other is to place the artifact in an HTML form control. When URL encoding is  
1053 used, the HTTP GET method is used to deliver the message, while POST is used with form encoding. All  
1054 endpoints that support this binding MUST support both techniques.

### 1055 **3.6.3.1 RelayState**

1056 RelayState data MAY be included with a SAML artifact transmitted with this binding. The value MUST  
1057 NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the message  
1058 independent of any other protections that may or may not exist during message transmission. Signing is  
1059 not realistic given the space limitation, but because the value is exposed to third-party tampering, the  
1060 entity SHOULD ensure that the value has not been tampered with by using a checksum, a pseudo-  
1061 random value, or similar means.

1062 If an artifact that represents a SAML request is accompanied by RelayState data, then the SAML  
1063 responder MUST return its SAML protocol response using a binding that also supports a RelayState  
1064 mechanism, and it MUST place the exact data it received with the artifact into the corresponding  
1065 RelayState parameter in the response.

1066 If no such value is included with an artifact representing a SAML request, or if the SAML response  
1067 message is being generated without a corresponding request, then the SAML responder MAY include  
1068 RelayState data to be interpreted by the recipient based on the use of a profile or prior agreement  
1069 between the parties.

### 1070 **3.6.3.2 URL Encoding**

1071 To encode an artifact into a URL, the artifact value is URL-encoded and placed in a query string  
1072 parameter named `SAMLart`.

1073 If a "RelayState" value is to accompany the SAML artifact, it MUST be URL-encoded and placed in an  
1074 additional query string parameter named `RelayState`.

### 1075 3.6.3.3 Form Encoding

1076 A SAML artifact is form-encoded by placing it in a hidden form control within a form as defined by  
1077 [HTML401], chapter 17. The HTML document MUST adhere to the XHTML specification, [XHTML]. The  
1078 form control MUST be named `SAMLart`. Any additional form controls or presentation MAY be included but  
1079 MUST NOT be required in order for the recipient to process the artifact.

1080 If a "RelayState" value is to accompany the SAML artifact, it MUST be placed in an additional hidden form  
1081 control named `RelayState`, within the same form with the SAML message.

1082 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using  
1083 this binding to which the artifact is to be delivered. The `method` attribute MUST be set to "POST".

1084 Any technique supported by the user agent MAY be used to cause the submission of the form, and any  
1085 form content necessary to support this MAY be included, such as submit controls and client-side scripting  
1086 commands. However, the recipient MUST be able to process the artifact without regard for the  
1087 mechanism by which the form submission is initiated.

1088 Note that any form control values included MUST be transformed so as to be safe to include in the  
1089 XHTML document. This includes transforming characters such as quotes into HTML entities, etc.

### 1090 3.6.4 Artifact Format

1091 With respect to this binding, an artifact is a short, opaque string. Different types can be defined and used  
1092 without affecting the binding. The important characteristics are the ability of an artifact receiver to identify  
1093 the issuer of the artifact, resistance to tampering and forgery, uniqueness, and compactness.

1094 The general format of any artifact includes a mandatory two-byte artifact type code and a two-byte index  
1095 value identifying a specific endpoint of the artifact resolution service of the issuer, as follows:

```
1096 SAML_artifact      := B64( TypeCode EndpointIndex RemainingArtifact )
1097 TypeCode           := Byte1Byte2
1098 EndpointIndex      := Byte1Byte2
```

1099 The notation `B64( TypeCode EndpointIndex RemainingArtifact )` stands for the application of  
1100 the base64 [RFC2045] transformation to the catenation of the `TypeCode`, `EndpointIndex`, and  
1101 `RemainingArtifact`.

1102 The following practices are RECOMMENDED for the creation of SAML artifacts:

- 1103 • Each issuer is assigned an identifying URI, also known as the issuer's entity (or provider) ID. See  
1104 Section 8.3.6 of [SAMLCore] for a discussion of this kind of identifier.
- 1105 • The issuer constructs the `SourceID` component of the artifact by taking the SHA-1 hash of the  
1106 identification URL. The hash value is NOT encoded into hexadecimal.
- 1107 • The `MessageHandle` value is constructed from a cryptographically strong random or  
1108 pseudorandom number sequence [RFC1750] generated by the issuer. The sequence consists of  
1109 values of at least 16 bytes in size. These values should be padded as needed to a total length of 20  
1110 bytes.

1111 The following describes the single artifact type defined by SAML V2.0. [\[E4\]Although the general artifact  
1112 structure resembles that used in prior versions of SAML and the type code of the single format described  
1113 below does not conflict with previously defined formats, there is explicitly no correspondence between  
1114 SAML V2.0 artifacts and those found in any previous specifications, and artifact formats not defined  
1115 specifically for use with SAML V2.0 MUST NOT be used with this binding.](#)

#### 1116 3.6.4.1 Required Information

1117 **Identification:** urn:oasis:names:tc:SAML:2.0:artifact-04

1118 **Contact information:** [security-services-comment@lists.oasis-open.org](mailto:security-services-comment@lists.oasis-open.org)

1119 **Description:** Given below.

1120 **Updates:** None.

### 1121 **3.6.4.2 Format Details**

1122 SAML V2.0 defines an artifact type of type code 0x0004. This artifact type is defined as follows:

```
1123     TypeCode           := 0x0004
1124     RemainingArtifact := SourceID MessageHandle
1125     SourceID          := 20-byte_sequence
1126     MessageHandle     := 20-byte_sequence
```

1127 `SourceID` is a 20-byte sequence used by the artifact receiver to determine artifact issuer identity and the  
1128 set of possible resolution endpoints.

1129 It is assumed that the destination site will maintain a table of `SourceID` values as well as one or more  
1130 indexed URL endpoints (or addresses) for the corresponding SAML responder. The SAML metadata  
1131 specification [SAMLMeta] MAY be used for this purpose. On receiving the SAML artifact, the receiver  
1132 determines if the `SourceID` belongs to a known artifact issuer and obtains the location of the SAML  
1133 responder using the `EndpointIndex` before sending a SAML `<samlp:ArtifactResolve>` message  
1134 to it.

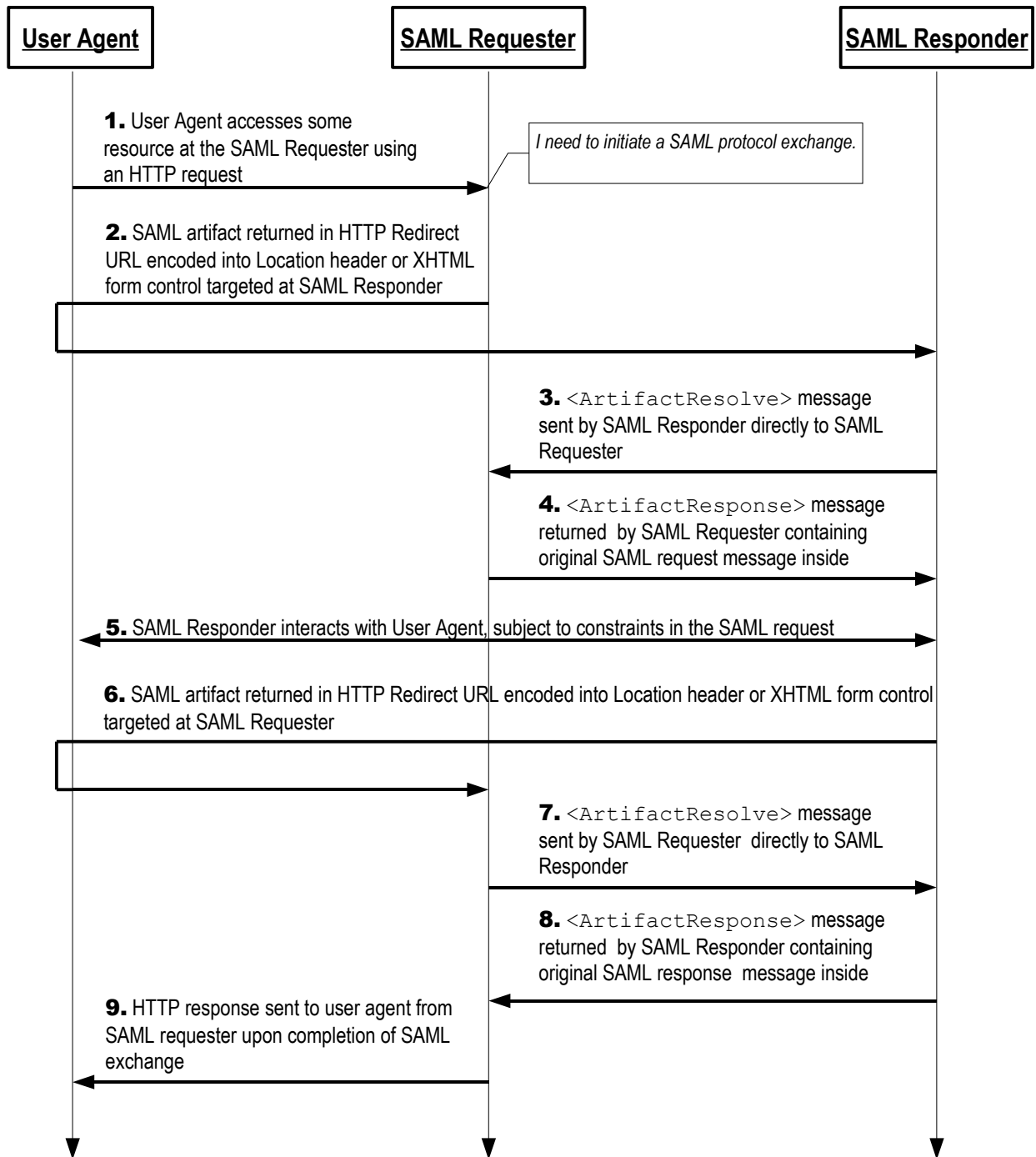
1135 Any two artifact issuers with a common receiver MUST use distinct `SourceID` values. Construction of  
1136 `MessageHandle` values is governed by the principle that they SHOULD have no predictable relationship  
1137 to the contents of the referenced message at the issuing site and it MUST be infeasible to construct or  
1138 guess the value of a valid, outstanding message handle.

### 1139 **3.6.5 Message Exchange**

1140 The system model used for SAML conversations by means of this binding is a request-response model in  
1141 which an artifact reference takes the place of the actual message content, and the artifact reference is  
1142 sent to the user agent in an HTTP response and delivered to the message recipient in an HTTP request.  
1143 The HTTP interactions before, between, and after these exchanges take place is unspecified. Both the  
1144 SAML requester and responder are assumed to be HTTP responders.

1145 Additionally, it is assumed that on receipt of an artifact by way of the user agent, the recipient invokes a  
1146 separate, direct exchange with the artifact issuer using the Artifact Resolution Protocol defined in  
1147 [SAMLCore]. This exchange MUST use a binding that does not use the HTTP user agent as an  
1148 intermediary, such as the SOAP binding. On the successful acquisition of a SAML protocol message, the  
1149 artifact is discarded and the processing of the primary SAML protocol exchange resumes (or ends, if the  
1150 message is a response).

1151 Issuing and delivering an artifact, along with the subsequent resolution step, constitutes half of the overall  
1152 SAML protocol exchange. This binding can be used to deliver either or both halves of a SAML protocol  
1153 exchange. A binding composable with it, such as the HTTP Redirect (see Section 3.4) or POST (see  
1154 Section 3.5) binding, MAY be used to carry the other half of the exchange. The following sequence  
1155 assumes that the artifact binding is used for both halves. See the diagram below illustrating the messages  
1156 exchanged.



- 1157 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of  
 1158 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 1159 2. The system entity acting as a SAML requester responds to an HTTP request from the user agent by  
 1160 returning an artifact representing a SAML request.
- 1161 • If URL-encoded, the artifact is returned encoded into the HTTP response's Location  
 1162 header, and the HTTP status MUST be either 303 or 302. The SAML requester MAY  
 1163 include additional presentation and content in the HTTP response to facilitate the user  
 1164 agent's transmission of the message, as defined in HTTP 1.1 [RFC2616]. The user



- 1165 agent delivers the artifact by issuing an HTTP GET request to the SAML responder.
- 1166 • If form-encoded, then the artifact is returned in an XHTML document containing the  
1167 form and content defined in Section 3.6.3.3. The user agent delivers the artifact by  
1168 issuing an HTTP POST request to the SAML responder.
- 1169 3. The SAML responder determines the SAML requester by examining the artifact (the exact process  
1170 depends on the type of artifact), and issues a `<samlp:ArtifactResolve>` request containing  
1171 the artifact to the SAML requester using a direct SAML binding, temporarily reversing roles.
- 1172 4. Assuming the necessary conditions are met, the SAML requester returns a  
1173 `<samlp:ArtifactResponse>` containing the original SAML request message it wishes the  
1174 SAML responder to process.
- 1175 5. In general, the SAML responder MAY respond to the SAML request by immediately returning a  
1176 SAML artifact or MAY return arbitrary content to facilitate subsequent interaction with the user agent  
1177 necessary to fulfill the request. Specific protocols and profiles may include mechanisms to indicate  
1178 the requester's level of willingness to permit this kind of interaction (for example, the `IsPassive`  
1179 attribute in `<samlp:AuthnRequest>`).
- 1180 6. Eventually the responder SHOULD return a SAML artifact to the user agent to be returned to the  
1181 SAML requester. The SAML response artifact is returned in the same fashion as described for the  
1182 SAML request artifact in step 2. The SAML requester determines the SAML responder by examining  
1183 the artifact, and issues a `<samlp:ArtifactResolve>` request containing the artifact to the SAML  
1184 responder using a [\[E31\]synchronousdirect](#) SAML binding, as in step 3.
- 1185 7. Assuming the necessary conditions are met, the SAML responder returns a  
1186 `<samlp:ArtifactResponse>` containing the SAML response message it wishes the requester to  
1187 process, as in step 4.
- 1188 8. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the  
1189 user agent.

### 1190 **3.6.5.1 HTTP and Caching Considerations**

1191 HTTP proxies and the user agent intermediary should not cache SAML artifacts. To ensure this, the  
1192 following rules SHOULD be followed.

1193 When returning SAML artifacts using HTTP 1.1, HTTP responders SHOULD:

- 1194 • Include a `Cache-Control` header field set to "no-cache, no-store".  
1195 • Include a `Pragma` header field set to "no-cache".

1196 There are no other restrictions on the use of HTTP headers.

### 1197 **3.6.5.2 Security Considerations**

1198 This binding uses a combination of indirect transmission of a message reference followed by a direct  
1199 exchange to return the actual message. As a result, the message reference (artifact) need not itself be  
1200 authenticated or integrity protected, but the callback request/response exchange that returns the actual  
1201 message MAY be mutually authenticated and integrity protected, depending on the environment of use.

1202 If the actual SAML protocol message is intended for a specific recipient, then the artifact's issuer MUST  
1203 authenticate the sender of the subsequent `<samlp:ArtifactResolve>` message before returning the  
1204 actual message.

1205 The transmission of an artifact to and from the user agent SHOULD be protected with confidentiality; SSL  
1206 3.0 [SSL3] or TLS 1.0 [RFC2246] SHOULD be used. The callback request/response exchange that  
1207 returns the actual message MAY be protected, depending on the environment of use.

1208 In general, this binding relies on the artifact as a hard-to-forge short-term reference and applies other

1209 security measures to the callback request/response that returns the actual message. All artifacts MUST  
1210 have a single-use semantic enforced by the artifact issuer.

1211 Furthermore, it is RECOMMENDED that artifact receivers also enforce a single-use semantic on the  
1212 artifact values they receive, to prevent an attacker from interfering with the resolution of an artifact by a  
1213 user agent and then resubmitting it to the artifact receiver. If an attempt to resolve an artifact does not  
1214 complete successfully, the artifact SHOULD be placed into a blocked artifact list for a period of time that  
1215 exceeds a reasonable acceptance period during which the artifact issuer would resolve the artifact.

1216 Note also that there is no mechanism defined to protect the integrity of the relationship between the  
1217 artifact and the "RelayState" value, if any. That is, an attacker can potentially recombine a pair of valid  
1218 HTTP responses by switching the "RelayState" values associated with each artifact. As a result, the  
1219 producer/consumer of "RelayState" information MUST take care not to associate sensitive state  
1220 information with the "RelayState" value without taking additional precautions (such as based on the  
1221 information in the SAML protocol message retrieved via artifact).

1222 [\[E59\]Finally, note that the use of the Destination attribute in the root SAML element of the protocol](#)  
1223 [message is unspecified by this binding, because of the message indirection involved.](#)

### 1224 3.6.6 Error Reporting

1225 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD  
1226 return a response message with a second-level <samlp:StatusCode> value of  
1227 urn:oasis:names:tc:SAML:2.0:status:RequestDenied.

1228 HTTP interactions during the message exchange MUST NOT use HTTP error status codes to indicate  
1229 failures in SAML processing, since the user agent is not a full party to the SAML protocol exchange.

1230 If the issuer of an artifact receives a <samlp:ArtifactResolve> message that it can understand, it  
1231 MUST return a <samlp:ArtifactResponse> with a <samlp:StatusCode> value of  
1232 urn:oasis:names:tc:SAML:2.0:status:Success, even if it does not return the corresponding  
1233 message (for example because the artifact requester is not authorized to receive the message or the  
1234 artifact is no longer valid).

1235 For more information about SAML status codes, see the SAML assertions and protocols specification  
1236 [SAMLCore].

### 1237 3.6.7 Metadata Considerations

1238 Support for [\[E2\]receiving messages using](#) the HTTP Artifact binding SHOULD be reflected by indicating  
1239 URL endpoints at which requests and responses for a particular protocol or profile should be sent. Either a  
1240 single endpoint or distinct request and response endpoints MAY be supplied. ~~One or more indexed~~  
1241 ~~endpoints for processing <samlp:ArtifactResolve> messages SHOULD also be described.~~ [Support](#)  
1242 [for sending messages using this binding SHOULD be accompanied by one or more indexed](#)  
1243 [<md:ArtifactResolutionService> endpoints for processing <samlp:ArtifactResolve>](#)  
1244 [messages.](#)

### 1245 3.6.8 Example SAML Message Exchange Using HTTP Artifact

1246 In this example, a <LogoutRequest> and <LogoutResponse> message pair is exchanged using the  
1247 HTTP Artifact binding, with the artifact resolution taking place using the SOAP binding bound to HTTP.

1248 First, here are the actual SAML protocol messages being exchanged:

```
1249 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
1250 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
1251 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
1252 21T19:00:49Z" Version="2.0">  
1253 <Issuer>https://IdentityProvider.com/SAML</Issuer>
```



```

1254     <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
1255 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>
1256     <samlp:SessionIndex>1</samlp:SessionIndex>
1257 </samlp:LogoutRequest>

1258 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1259 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1260 ID="b0730d21b628110d8b7e004005b13a2b"
1261 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"
1262 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">
1263 <Issuer>https://ServiceProvider.com/SAML</Issuer>
1264 <samlp:Status>
1265 <samlp:StatusCode
1266 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1267 </samlp:Status>
1268 </samlp:LogoutResponse>

```

1269 The initial HTTP request from the user agent in step 1 is not defined by this binding. To initiate the logout  
1270 protocol exchange, the SAML requester returns the following HTTP response, containing a SAML artifact.  
1271 Note that the line feeds in the HTTP Location header below are a result of document formatting, and  
1272 there are no line feeds in the actual header value.

```

1273 HTTP/1.1 302 Object Moved
1274 Date: 21 Jan 2004 07:00:49 GMT
1275 Location: https://ServiceProvider.com/SAML/SLO/Browser?
1276 SAMLart=AAQAADWNEw5VT47wcO4zX%2FiEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU
1277 %3D&RelayState=0043bfc1bc45110dae17004005b13a2b
1278 Content-Type: text/html; charset=iso-8859-1

```

1279 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact  
1280 Resolution protocol and the SOAP binding in steps 3 and 4, as follows:

1281 Step 3:

```

1282 POST /SAML/Artifact/Resolve HTTP/1.1
1283 Host: IdentityProvider.com
1284 Content-Type: text/xml
1285 Content-Length: nnn
1286 SOAPAction: http://www.oasis-open.org/committees/security
1287 <SOAP-ENV:Envelope
1288 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1289 <SOAP-ENV:Body>
1290 <samlp:ArtifactResolve
1291 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1292 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1293 ID="_6c3a4f8b9c2d" Version="2.0"
1294 IssueInstant="2004-01-21T19:00:49Z">
1295 <Issuer>https://ServiceProvider.com/SAML</Issuer>
1296 <Artifact>
1297 AAQAADWNEw5VT47wcO4zX/iEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU=
1298 </Artifact>
1299 </samlp:ArtifactResolve>
1300 </SOAP-ENV:Body>
1301 </SOAP-ENV:Envelope>

```

1302 Step 4:

```

1303 HTTP/1.1 200 OK
1304 Date: 21 Jan 2004 07:00:49 GMT
1305 Content-Type: text/xml
1306 Content-Length: nnnn

1307 <SOAP-ENV:Envelope
1308 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1309 <SOAP-ENV:Body>
1310 <samlp:ArtifactResponse
1311 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1312 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"

```

```

1313         ID="_FQvGknDfws2Z" Version="2.0"
1314         InResponseTo=" 6c3a4f8b9c2d"
1315         IssueInstant="2004-01-21T19:00:49Z">
1316         <Issuer>https://IdentityProvider.com/SAML</Issuer>
1317         <samlp:Status>
1318             <samlp:StatusCode
1319                 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1320             </samlp:Status>
1321             <samlp:LogoutRequest ID="d2b7c388cec36fa7c39c28fd298644a8"
1322                 IssueInstant="2004-01-21T19:00:49Z"
1323                 Version="2.0">
1324                 <Issuer>https://IdentityProvider.com/SAML</Issuer>
1325                 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
1326 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>
1327                 <samlp:SessionIndex>1</samlp:SessionIndex>
1328             </samlp:LogoutRequest>
1329         </samlp:ArtifactResponse>
1330     </SOAP-ENV:Body>
1331 </SOAP-ENV:Envelope>

```

1332 After any unspecified interactions may have taken place, the SAML responder returns a second SAML  
1333 artifact in its HTTP response in step 6:

```

1334 HTTP/1.1 302 Object Moved
1335 Date: 21 Jan 2004 07:05:49 GMT
1336 Location: https://IdentityProvider.com/SAML/SLO/Response?
1337 SAMLart=AAQAAFgIZXv5%2BQaBaE5qYurHWJ0lnAgLAsqfnyidHIggbFU0mlSGFTyQiPc
1338 %3D&RelayState=0043bfclbc45110dae17004005b13a2b
1339 Content-Type: text/html; charset=iso-8859-1

```

1340 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact  
1341 Resolution protocol and the SOAP binding in steps 7 and 8, as follows:

1342 Step 7:

```

1343 POST /SAML/Artifact/Resolve HTTP/1.1
1344 Host: ServiceProvider.com
1345 Content-Type: text/xml
1346 Content-Length: nnn
1347 SOAPAction: http://www.oasis-open.org/committees/security
1348 <SOAP-ENV:Envelope
1349     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1350     <SOAP-ENV:Body>
1351         <samlp:ArtifactResolve
1352             xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1353             xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1354             ID="_ec36fa7c39" Version="2.0"
1355             IssueInstant="2004-01-21T19:05:49Z">
1356             <Issuer>https://IdentityProvider.com/SAML</Issuer>
1357             <Artifact>
1358                 AAQAAFgIZXv5+QaBaE5qYurHWJ0lnAgLAsqfnyidHIggbFU0mlSGFTyQiPc=
1359             </Artifact>
1360         </samlp:ArtifactResolve>
1361     </SOAP-ENV:Body>
1362 </SOAP-ENV:Envelope>

```

1363 Step 8:

```

1364 HTTP/1.1 200 OK
1365 Date: 21 Jan 2004 07:05:49 GMT
1366 Content-Type: text/xml
1367 Content-Length: nnnn
1368 <SOAP-ENV:Envelope
1369     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1370     <SOAP-ENV:Body>
1371         <samlp:ArtifactResponse
1372             xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1373             xmlns="urn:oasis:names:tc:SAML:2.0:assertion"

```

```

1374 ID="_FQvGknDfws2Z" Version="2.0"
1375 InResponseTo="_ec36fa7c39"
1376 IssueInstant="2004-01-21T19:05:49Z">
1377 <Issuer>https://ServiceProvider.com/SAML</Issuer>
1378 <samlp:Status>
1379   <samlp:StatusCode
1380     Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1381   </samlp:Status>
1382   <samlp:LogoutResponse ID="_b0730d21b628110d8b7e004005b13a2b"
1383     InResponseTo="_d2b7c388cec36fa7c39c28fd298644a8"
1384     IssueInstant="2004-01-21T19:05:49Z"
1385     Version="2.0">
1386     <Issuer>https://ServiceProvider.com/SAML</Issuer>
1387     <samlp:Status>
1388       <samlp:StatusCode
1389         Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1390       </samlp:Status>
1391     </samlp:LogoutResponse>
1392   </samlp:ArtifactResponse>
1393 </SOAP-ENV:Body>
1394 </SOAP-ENV:Envelope>

```

## 1395 3.7 SAML URI Binding

1396 URIs are a protocol-independent means of referring to a resource. This binding is not a general SAML  
 1397 request/response binding, but rather supports the encapsulation of a <samlp:AssertionIDRequest>  
 1398 message with a single <saml:AssertionIDRef> into the resolution of a URI. The result of a successful  
 1399 request is a SAML <saml:Assertion> element (but not a complete SAML response).

1400 Like SOAP, URI resolution can occur over multiple underlying transports. This binding has  
 1401 [\[E24\]protocol/transport-independent aspects](#), but also calls out [as mandatory](#) the [implementation of HTTP](#)  
 1402 [URI](#) [use of HTTP with SSL 3.0 \[SSL3\] or TLS 1.0 \[RFC2246\]](#) as REQUIRED (mandatory to implement).

### 1403 3.7.1 Required Information

1404 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:URI

1405 **Contact information:** [security-services-comment@lists.oasis-open.org](mailto:security-services-comment@lists.oasis-open.org)

1406 **Description:** Given below.

1407 **Updates:** None

### 1408 3.7.2 Protocol-Independent Aspects of the SAML URI Binding

1409 The following sections define aspects of the SAML URI binding that are independent of the underlying  
 1410 transport protocol of the URI resolution process.

#### 1411 3.7.2.1 Basic Operation

1412 A SAML URI reference identifies a specific SAML assertion. The result of resolving the URI MUST be a  
 1413 message containing the assertion, or a transport-specific error. The specific format of the message  
 1414 depends on the underlying transport protocol. If the transport protocol permits the returned content to be  
 1415 described, such as HTTP 1.1 [RFC2616], then the assertion MAY be encoded in whatever format is  
 1416 permitted. If not, the assertion MUST be returned in a form which can be unambiguously interpreted as or  
 1417 transformed into an XML serialization of the assertion.

1418 It MUST be the case that if the same URI reference is resolved in the future, then either the same SAML  
 1419 assertion, or an error, is returned. That is, the reference MAY be persistent but MUST consistently  
 1420 reference the same assertion, if any.

### 1421 **3.7.3 Security Considerations**

1422 Indirect use of a SAML assertion presents dangers if the binding of the reference to the result is not  
1423 secure. The particular threats and their severity depend on the use to which the assertion is being put. In  
1424 general, the result of resolving a URI reference to a SAML assertion SHOULD only be trusted if the  
1425 requester can be certain of the identity of the responder and that the contents have not been modified in  
1426 transit.

1427 It is often not sufficient that the assertion itself be signed, because URI references are by their nature  
1428 somewhat opaque to the requester. The requester SHOULD have independent means to ensure that the  
1429 assertion returned is actually the one that is represented by the URI; this is accomplished by both  
1430 authenticating the responder and relying on the integrity of the response.

### 1431 **3.7.4 MIME Encapsulation**

1432 For resolution protocols that support MIME as a content description and packaging mechanism, the  
1433 resulting assertion SHOULD be returned as a MIME entity of type `application/samlassertion+xml`,  
1434 as defined by [SAMLmime].

### 1435 **3.7.5 Use of HTTP URIs**

1436 A SAML authority that claims conformance to the SAML URI binding MUST implement support for HTTP.  
1437 This section describes certain specifics of using HTTP URIs, including URI syntax, HTTP headers, and  
1438 error reporting.

#### 1439 **3.7.5.1 URI Syntax**

1440 In general, there are no restrictions on the permissible syntax of a SAML URI reference as long as the  
1441 SAML authority responsible for the reference creates the message containing it. However, authorities  
1442 MUST support a URL endpoint at which an HTTP request can be sent with a single query string  
1443 parameter named `ID`. There MUST be no query string in the endpoint URL itself independent of this  
1444 parameter.

1445 For example, if the documented endpoint at an authority is "https://saml.example.edu/assertions", a  
1446 request for an assertion with an `ID` of `abcde` can be sent to:

1447 `https://saml.example.edu/assertions?ID=abcde`

1448 Note that [E31]the URI syntax does not support the use of wildcards ~~is not allowed for on~~ such `ID`-queries.

#### 1449 **3.7.5.2 HTTP and Caching Considerations**

1450 HTTP proxies MUST NOT cache SAML assertions. To ensure this, the following rules SHOULD be  
1451 followed.

1452 When returning SAML assertions using HTTP 1.1, HTTP responders SHOULD:

- 1453 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 1454 • Include a `Pragma` header field set to "no-cache".

#### 1455 **3.7.5.3 Security Considerations**

1456 RFC 2617 [RFC2617] describes possible attacks in the HTTP environment when basic or message-digest  
1457 authentication schemes are used.

1458 Use of SSL 3.0 [SSL3] or TLS 1.0 [RFC2246] is STRONGLY RECOMMENDED as a means of  
1459 authentication, integrity protection, and confidentiality.

#### 1460 **3.7.5.4 Error Reporting**

1461 As an HTTP protocol exchange, the appropriate HTTP status code SHOULD be used to indicate the result  
1462 of a request. For example, a SAML responder that refuses to perform a message exchange with the  
1463 SAML requester SHOULD return a "403 Forbidden" response. If the assertion specified is unknown to  
1464 the responder, then a "404 Not Found" response SHOULD be returned. In these cases, the content of  
1465 the HTTP body is not significant.

#### 1466 **3.7.5.5 Metadata Considerations**

1467 Support for the URI binding over HTTP SHOULD be reflected by indicating a URL endpoint at which  
1468 requests for arbitrary assertions are to be sent.

#### 1469 **3.7.5.6 Example SAML Message Exchange Using an HTTP URI**

1470 Following is an example of a request for an assertion.

```
1471 GET /SamlService?ID=abcde HTTP/1.1  
1472 Host: www.example.com
```

1473 Following is an example of the corresponding response, which supplies the requested assertion.

```
1474 HTTP/1.1 200 OK  
1475 Content-Type: application/samlassertion+xml  
1476 Cache-Control: no-cache, no-store  
1477 Pragma: no-cache  
1478 Content-Length: nnnn  
  
1479 <saml:Assertion ID="abcde" ...>  
1480 ...  
1481 </saml:Assertion>
```

---

## 4 References

1482

- 1483 [HTML401] D. Raggett et al. *HTML 4.01 Specification*. World Wide Web Consortium  
1484 Recommendation, December 1999. See <http://www.w3.org/TR/html4>.
- 1485 [Liberty] The Liberty Alliance Project. See <http://www.projectliberty.org>.
- 1486 [PAOS] R. Aarts. *Liberty Reverse HTTP Binding for SOAP Specification Version 1.0*.  
1487 Liberty Alliance Project, 2003. See [https://www.projectliberty.org/specs/liberty-](https://www.projectliberty.org/specs/liberty-paos-v1.0.pdf)  
1488 [paos-v1.0.pdf](https://www.projectliberty.org/specs/liberty-paos-v1.0.pdf).
- 1489 [RFC1750] D. Eastlake et al. *Randomness Recommendations for Security*. IETF RFC 1750,  
1490 December 1994. See <http://www.ietf.org/rfc/rfc1750.txt>.
- 1491 [RFC2045] N. Freed et al. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of*  
1492 *Internet Message Bodies*, IETF RFC 2045, November 1996. See  
1493 <http://www.ietf.org/rfc/rfc2045.txt>.
- 1494 [RFC2119] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF  
1495 RFC 2119, March 1997. See <http://www.ietf.org/rfc/rfc2119.txt>.
- 1496 [RFC2246] T. Dierks et al. *The TLS Protocol Version 1.0*. IETF RFC 2246, January 1999.  
1497 See <http://www.ietf.org/rfc/rfc2246.txt>.
- 1498 [RFC2279] F. Yergeau. *UTF-8, a transformation format of ISO 10646*. IETF RFC 2279,  
1499 January 1998. See <http://www.ietf.org/rfc/rfc2279.txt>.
- 1500 [RFC2616] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC 2616, June  
1501 1999. See <http://www.ietf.org/rfc/rfc2616.txt>.
- 1502 [RFC2617] J. Franks et al. *HTTP Authentication: Basic and Digest Access Authentication*.  
1503 IETF RFC 2617, June 1999. See <http://www.ietf.org/rfc/rfc2617.txt>.
- 1504 [SAML11Bind] E. Maler et al. *Bindings and Profiles for the OASIS Security Assertion Markup*  
1505 *Language (SAML)*. OASIS, September 2003. Document ID oasis-sstc-saml-  
1506 bindings-1.1. See <http://www.oasis-open.org/committees/security/>.
- 1507 [SAMLConform] P. Mishra et al. *Conformance Requirements for the OASIS Security Assertion*  
1508 *Markup Language (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-  
1509 conformance-2.0-os. See <http://www.oasis-open.org/committees/security/>.
- 1510 [SAMLCore] S. Cantor et al. *Assertions and Protocols for the OASIS Security Assertion*  
1511 *Markup Language (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-  
1512 core-2.0-os. See <http://www.oasis-open.org/committees/security/>.
- 1513 [SAMLGloss] J. Hodges et al. *Glossary for the OASIS Security Assertion Markup Language*  
1514 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-glossary-2.0-os.  
1515 See <http://www.oasis-open.org/committees/security/>.
- 1516 [SAMLMeta] S. Cantor et al. *Metadata for the OASIS Security Assertion Markup Language*  
1517 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-metadata-2.0-os.  
1518 See <http://www.oasis-open.org/committees/security/>.
- 1519 [SAMLmime] ~~[E57]application/saml+xml Media Type Registration, IETF Internet Draft,~~  
1520 ~~<http://www.ietf.org/internet-drafts/draft-hodges-saml-mediatype-01.txt>.~~ OASIS  
1521 Security Services Technical Committee (SSTC). "application/samlassertion+xml  
1522 MIME Media Type Registration", IANA MIME Media Types Registry  
1523 application/samlassertion+xml, December 2004. See  
1524 <http://www.iana.org/assignments/media-types/application/samlassertion+xml>.
- 1525 [SAMLProfile] S. Cantor et al. *Profiles for the OASIS Security Assertion Markup Language*  
1526 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-profiles-2.0-os. See  
1527 <http://www.oasis-open.org/committees/security/>.
- 1528 [SAMLSecure] F. Hirsch et al. *Security and Privacy Considerations for the OASIS Security*

1529		<i>Assertion Markup Language (SAML) V2.0</i> . OASIS SSTC, March 2005. Document ID saml-sec-consider-2.0-os. See <a href="http://www.oasis-open.org/committees/security/">http://www.oasis-open.org/committees/security/</a> .
1530		
1531		
1532	<b>[SOAP11]</b>	D. Box et al. <i>Simple Object Access Protocol (SOAP) 1.1</i> . World Wide Web Consortium Note, May 2000. See <a href="http://www.w3.org/TR/2000/NOTE-SOAP-20000508/">http://www.w3.org/TR/2000/NOTE-SOAP-20000508/</a> .
1533		
1534		
1535	<b>[SOAP-PRIMER]</b>	N. Mitra. <i>SOAP Version 1.2 Part 0: Primer</i> . World Wide Web Consortium Recommendation, June 2003. See <a href="http://www.w3.org/TR/soap12-part0/">http://www.w3.org/TR/soap12-part0/</a> ,
1536		
1537	<b>[SSL3]</b>	A. Frier et al. <i>The SSL 3.0 Protocol</i> . Netscape Communications Corp, November 1996.
1538		
1539	<b>[SSTCWeb]</b>	OASIS Security Services Technical Committee website, <a href="http://www.oasis-open.org/committees/security/">http://www.oasis-open.org/committees/security/</a> .
1540		
1541	<b>[XHTML]</b>	<i>XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)</i> . World Wide Web Consortium Recommendation, August 2002. See <a href="http://www.w3.org/TR/xhtml1/">http://www.w3.org/TR/xhtml1/</a> .
1542		
1543		
1544	<b>[XMLSig]</b>	D. Eastlake et al. <i>XML-Signature Syntax and Processing</i> . <del>[E74]</del> <i>Second Edition</i> . World Wide Web Consortium Recommendation, <del>June 2008</del> <i>February 2002</i> . See <a href="http://www.w3.org/TR/xmlsig-core/">http://www.w3.org/TR/xmlsig-core/</a> .
1545		
1546		



---

1547 **Appendix A. Registration of MIME media type**  
1548 **application/samlassertion+xml**

1549 **Introduction**

1550 This document defines a MIME media type -- `application/samlassertion+xml` -- for use  
1551 with the XML serialization of SAML (Security Assertion Markup Language) assertions.

1552 The SAML specification sets -- [SAMLv1.0], [SAMLv1.1], [SAMLv2.0] -- are work products of the  
1553 OASIS Security Services Technical Committee [SSTC]. The SAML specifications define XML-  
1554 based constructs with which one may make, and convey, security assertions. Using SAML, one  
1555 can assert that an authentication event pertaining to some subject has occurred and convey said  
1556 assertion to a relying party, for example.

1557 SAML assertions, which are explicitly versioned, are defined by [SAMLv1Core], [SAMLv11Core],  
1558 and [SAMLv2Core].

1559 **MIME media type name**

1560 `application`

1561 **MIME subtype name**

1562 `samlassertion+xml`

1563 **Required parameters**

1564 None

1565 **Optional parameters**

1566 `charset`

1567 Same as `charset` parameter of `application/xml` [RFC3023].

1568 **Encoding considerations**

1569 Same as for `application/xml` [RFC3023].

1570 **Security considerations**

1571 Per their specification, `samlassertion+xml`-typed objects do not contain executable content.  
1572 However, SAML assertions are XML-based objects [XML]. As such, they have all of the general  
1573 security considerations presented in Section 10 of [RFC3023], as well as additional ones, since  
1574 they are explicit security objects. For example, `samlassertion+xml`-typed objects will often  
1575 contain data that may identify or pertain to a natural person, and may be used as a basis for  
1576 sessions and access control decisions.

1577 To counter potential issues, `samlassertion+xml`-typed objects contain data that should be  
1578 signed appropriately by the sender. Any such signature must be verified by the recipient of the  
1579 data - both as a valid signature, and as being the signature of the sender. Issuers of  
1580 `samlassertion+xml`-typed objects containing SAMLv2 assertions may also encrypt all, or  
1581 portions of, the assertions (see [SAMLv2Core]).



1582 In addition, SAML profiles and protocol bindings specify use of secure channels as appropriate.

1583 [SAMLv2.0] incorporates various privacy-protection techniques in its design. For example: opaque  
1584 handles, specific to interactions between specific system entities, may be assigned to subjects.  
1585 The handles are mappable to wider-context identifiers (e.g. email addresses, account identifiers,  
1586 etc) by only the specific parties.

1587 For a more detailed discussion of SAML security considerations and specific security-related  
1588 design techniques, please refer to the SAML specifications listed in the below bibliography. The  
1589 specifications containing security-specific information have been explicitly listed for each version  
1590 of SAML.

## 1591 **Interoperability considerations**

1592 SAML assertions are explicitly versioned. Relying parties should ensure that they observe  
1593 assertion version information and behave accordingly. See chapters on SAML Versioning in  
1594 [SAMLv1Core], [SAMLv11Core], or [SAMLv2Core], as appropriate.

## 1595 **Published specification**

1596 [SAMLv2Bind] explicitly specifies use of the `application/samlassertion+xml` MIME media  
1597 type. However, it is conceivable that non-SAMLv2 assertions (i.e., SAMLv1 and/or SAMLv1.1)  
1598 might in practice be conveyed using SAMLv2 bindings.

## 1599 **Applications which use this media type**

1600 Potentially any application implementing SAML, as well as those applications implementing  
1601 specifications based on SAML, e.g. those available from the Liberty Alliance [LAP].

## 1602 **Additional information**

### 1603 **Magic number(s)**

1604 In general, the same as for `application/xml` [RFC3023]. In particular, the XML root element of the  
1605 returned object will have a namespace-qualified name with:

- 1606 – a local name of: `Assertion`
- 1607 – a namespace URI of: one of the version-specific SAML assertion XML  
1608 namespace URIs, as defined by the appropriate version-specific SAML "core"  
1609 specification (see bibliography).

1610 With SAMLv2.0 specifically, the root element of the returned object may be either  
1611 `<saml:Assertion>` or `<saml:EncryptedAssertion>`, where "saml" represents any XML  
1612 namespace prefix that maps to the SAMLv2.0 assertion namespace URI:

1613 `urn:oasis:names:tc:SAML:2.0:assertion`

### 1614 **File extension(s)**

1615 None

### 1616 **Macintosh File Type Code(s)**

1617 None

## 1618 Person & email address to contact for further information

1619 This registration is made on behalf of the OASIS Security Services Technical Committee (SSTC)  
1620 Please refer to the SSTC website for current information on committee chairperson(s) and their  
1621 contact addresses: <http://www.oasis-open.org/committees/security/>. Committee members should  
1622 submit comments and potential errata to the security-services@lists.oasis-open.org list. Others  
1623 should submit them by filling out the web form located at [http://www.oasis-  
1624 open.org/committees/comments/form.php?wg\\_abbrev=security](http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=security).

1625 Additionally, the SAML developer community email distribution list, [saml-dev@lists.oasis-  
1626 open.org](mailto:saml-dev@lists.oasis-open.org), may be employed to discuss usage of the `application/samlassertion+xml`  
1627 MIME media type. The "saml-dev" mailing list is publicly archived here: [http://lists.oasis-  
1628 open.org/archives/saml-dev/](http://lists.oasis-open.org/archives/saml-dev/). To post to the "saml-dev" mailing list, one must subscribe to it. To  
1629 subscribe, send a message with the single word "subscribe" in the message body, to: [saml-dev-  
1630 request@lists.oasis-open.org](mailto:saml-dev-request@lists.oasis-open.org).

## 1631 Intended usage

1632 COMMON

## 1633 Author/Change controller

1634 The SAML specification sets are a work product of the OASIS Security Services Technical  
1635 Committee (SSTC). OASIS and the SSTC have change control over the SAML specification sets.

## 1636 Bibliography

- 1637 [LAP] *"Liberty Alliance Project"*. See <http://www.projectliberty.org/>
- 1638 [OASIS] *"Organization for the Advancement of Structured Information Systems"*.  
1639 See <http://www.oasis-open.org/>
- 1640 [RFC3023] M. Murata, S. St.Laurent, D. Kohn, "XML Media Types", IETF Request for  
1641 Comments 3023, January 2001. Available as [http://www.rfc-  
1642 editor.org/rfc/rfc3023.txt](http://www.rfc-editor.org/rfc/rfc3023.txt)
- 1643 [SAMLv1.0] OASIS Security Services Technical Committee, "Security Assertion  
1644 Markup Language (SAML) Version 1.0 Specification Set". OASIS  
1645 Standard 200205, November 2002. Available as [http://www.oasis-  
1646 open.org/committees/download.php/2290/oasis-sstc-saml-1.0.zip](http://www.oasis-open.org/committees/download.php/2290/oasis-sstc-saml-1.0.zip)
- 1647 [SAMLv1Bind] Prateek Mishra et al., "Bindings and Profiles for the OASIS Security  
1648 Assertion Markup Language (SAML)", OASIS, November 2002.  
1649 Document ID oasis-sstc-saml-bindings-1.0. See [http://www.oasis-  
1650 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1651 [SAMLv1Core] Phillip Hallam-Baker et al., "Assertions and Protocol for the OASIS  
1652 Security Assertion Markup Language (SAML)", OASIS, November 2002.  
1653 Document ID oasis-sstc-saml-core-1.0. See [http://www.oasis-  
1654 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1655 [SAMLv1Sec] Chris McLaren et al., "Security Considerations for the OASIS Security  
1656 Assertion Markup Language (SAML)", OASIS, November 2002.  
1657 Document ID oasis-sstc-saml-sec-consider-1.0. See [http://www.oasis-  
1658 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1659 [SAMLv1.1] OASIS Security Services Technical Committee, "Security Assertion  
1660 Markup Language (SAML) Version 1.1 Specification Set". OASIS  
1661 Standard 200308, August 2003. Available as [http://www.oasis-  
1662 open.org/committees/download.php/3400/oasis-sstc-saml-1.1-pdf-xsd.zip](http://www.oasis-open.org/committees/download.php/3400/oasis-sstc-saml-1.1-pdf-xsd.zip)
- 1663 [SAMLv11Bind] E. Maler et al. "Bindings and Profiles for the OASIS Security Assertion  
1664 Markup Language (SAML)". OASIS, September 2003. Document ID

1665		oasis-sstc-saml-bindings-1.1. <a href="http://www.oasis-open.org/committees/security/">http://www.oasis-open.org/committees/security/</a>
1666		
1667	[SAMLv11Core]	E. Maler et al. "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)". OASIS, September 2003. Document ID oasis-sstc-saml-core-1.1. <a href="http://www.oasis-open.org/committees/security/">http://www.oasis-open.org/committees/security/</a>
1668		
1669		
1670	[SAMLv11Sec]	E. Maler et al. "Security Considerations for the OASIS Security Assertion Markup Language (SAML)". OASIS, September 2003. Document ID oasis-sstc-saml-sec-consider-1.1. <a href="http://www.oasis-open.org/committees/security/">http://www.oasis-open.org/committees/security/</a>
1671		
1672		
1673		
1674	[SAMLv2.0]	OASIS Security Services Technical Committee, "Security Assertion Markup Language (SAML) Version 2.0 Specification Set". OASIS Standard, 15-Mar-2005. Available at: <a href="http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip">http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip</a>
1675		
1676		
1677		
1678	[SAMLv2Bind]	S. Cantor et al., "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-bindings-2.0-os. Available at: <a href="http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf">http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf</a>
1679		
1680		
1681		
1682	[SAMLv2Core]	S. Cantor et al., "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-core-2.0-os. Available at: <a href="http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf">http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf</a>
1683		
1684		
1685		
1686	[SAMLv2Prof]	S. Cantor et al., "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-profiles-2.0-os. Available at: <a href="http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf">http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf</a>
1687		
1688		
1689		
1690	[SAMLv2Sec]	F. Hirsch et al., "Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-sec-consider-2.0-os. Available at: <a href="http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf">http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf</a>
1691		
1692		
1693		
1694	[SSTC]	"OASIS Security Services Technical Committee". See <a href="http://www.oasis-open.org/committees/security/">http://www.oasis-open.org/committees/security/</a>
1695		
1696	[XML]	Bray, T., Paoli, J., Sperberg-McQueen, C.M. and E. Maler, François Yergeau, "Extensible Markup Language (XML) 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml, Feb 2004, Available as <a href="http://www.w3.org/TR/REC-xml/">http://www.w3.org/TR/REC-xml/</a>
1697		
1698		
1699		

---

## 1700 Appendix B. Acknowledgments

1701 The editors would like to acknowledge the contributions of the OASIS Security Services Technical  
1702 Committee, whose voting members at the time of publication were:

- 1703 • Conor Cahill, AOL
- 1704 • John Hughes, Atos Origin
- 1705 • Hal Lockhart, BEA Systems
- 1706 • Mike Beach, Boeing
- 1707 • Rebekah Metz, Booz Allen Hamilton
- 1708 • Rick Randall, Booz Allen Hamilton
- 1709 • Ronald Jacobson, Computer Associates
- 1710 • Gavenraj Sodhi, Computer Associates
- 1711 • Thomas Wisniewski, Entrust
- 1712 • Carolina Canales-Valenzuela, Ericsson
- 1713 • Dana Kaufman, Forum Systems
- 1714 • Irving Reid, Hewlett-Packard
- 1715 • Guy Denton, IBM
- 1716 • Heather Hinton, IBM
- 1717 • Maryann Hondo, IBM
- 1718 • Michael McIntosh, IBM
- 1719 • Anthony Nadalin, IBM
- 1720 • Nick Ragouzis, Individual
- 1721 • Scott Cantor, Internet2
- 1722 • Bob Morgan, Internet2
- 1723 • Peter Davis, Neustar
- 1724 • Jeff Hodges, Neustar
- 1725 • Frederick Hirsch, Nokia
- 1726 • Senthil Sengodan, Nokia
- 1727 • Abbie Barbir, Nortel Networks
- 1728 • Scott Kiester, Novell
- 1729 • Cameron Morris, Novell
- 1730 • Paul Madsen, NTT
- 1731 • Steve Anderson, OpenNetwork
- 1732 • Ari Kermaier, Oracle
- 1733 • Vamsi Motukuru, Oracle
- 1734 • Darren Platt, Ping Identity
- 1735 • Prateek Mishra, Principal Identity
- 1736 • Jim Lien, RSA Security
- 1737 • John Linn, RSA Security
- 1738 • Rob Philpott, RSA Security
- 1739 • Dipak Chopra, SAP
- 1740 • Jahan Moreh, Sigaba
- 1741 • Bhavna Bhatnagar, Sun Microsystems

- 1742 • Eve Maler, Sun Microsystems
- 1743 • Ronald Monzillo, Sun Microsystems
- 1744 • Emily Xu, Sun Microsystems
- 1745 • Greg Whitehead, Trustgenix

1746 The editors also would like to acknowledge the following former SSTC members for their contributions to  
1747 this or previous versions of the OASIS Security Assertions Markup Language Standard:

- 1748 • Stephen Farrell, Baltimore Technologies
- 1749 • David Orchard, BEA Systems
- 1750 • Krishna Sankar, Cisco Systems
- 1751 • Zahid Ahmed, CommerceOne
- 1752 • Tim Alsop, CyberSafe Limited
- 1753 • Carlisle Adams, Entrust
- 1754 • Tim Moses, Entrust
- 1755 • Nigel Edwards, Hewlett-Packard
- 1756 • Joe Pato, Hewlett-Packard
- 1757 • Bob Blakley, IBM
- 1758 • Marlena Erdos, IBM
- 1759 • Marc Chanliau, Netegrity
- 1760 • Chris McLaren, Netegrity
- 1761 • Lynne Rosenthal, NIST
- 1762 • Mark Skall, NIST
- 1763 • Charles Knouse, Oblix
- 1764 • Simon Godik, Overxeer
- 1765 • Charles Norwood, SAIC
- 1766 • Evan Prodromou, Securant
- 1767 • Robert Griffin, RSA Security (former editor)
- 1768 • Sai Allarvarpu, Sun Microsystems
- 1769 • Gary Ellison, Sun Microsystems
- 1770 • Chris Ferris, Sun Microsystems
- 1771 • Mike Myers, Traceroute Security
- 1772 • Phillip Hallam-Baker, VeriSign (former editor)
- 1773 • James Vanderbeek, Vodafone
- 1774 • Mark O'Neill, Vordel
- 1775 • Tony Palmer, Vordel

1776 Finally, the editors wish to acknowledge the following people for their contributions of material used as  
1777 input to the OASIS Security Assertions Markup Language specifications:

- 1778 • Thomas Gross, IBM
- 1779 • Birgit Pfitzmann, IBM

1780 The editors also would like to gratefully acknowledge Jahan Moreh of Sigaba, who during his tenure on  
1781 the SSTC was the primary editor of the errata working document and who made major substantive  
1782 contributions to all of the errata materials.

---

1783 **Appendix C. Notices**

1784 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that  
1785 might be claimed to pertain to the implementation or use of the technology described in this document or  
1786 the extent to which any license under such rights might or might not be available; neither does it represent  
1787 that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to  
1788 rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made  
1789 available for publication and any assurances of licenses to be made available, or the result of an attempt  
1790 made to obtain a general license or permission for the use of such proprietary rights by implementors or  
1791 users of this specification, can be obtained from the OASIS Executive Director.

1792 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or  
1793 other proprietary rights which may cover technology that may be required to implement this specification.  
1794 Please address the information to the OASIS Executive Director.

1795 **Copyright © OASIS Open 2005. All Rights Reserved.**

1796 This document and translations of it may be copied and furnished to others, and derivative works that  
1797 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and  
1798 distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and  
1799 this paragraph are included on all such copies and derivative works. However, this document itself may  
1800 not be modified in any way, such as by removing the copyright notice or references to OASIS, except as  
1801 needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights  
1802 defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it  
1803 into languages other than English.

1804 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors  
1805 or assigns.

1806 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
1807 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY  
1808 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR  
1809 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.