



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)

Document identifier: draft-sstc-core-~~3234~~

Location: <http://www.oasis-open.org/committees/security/docs>

Publication date: ~~April 10th 2002~~ April 4rd 2002

Maturity Level: Committee Working Draft

Send comments to: security-requestors-comment@lists.oasis-open.org

Note: Before sending a message to this list you must first subscribe; send an email message to security-requestors-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

Editors:

Phillip Hallam-Baker, VeriSign,
Eve Maler, Sun Microsystems

15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42

ASSERTIONS AND PROTOCOL FOR THE OASIS SECURITY ASSERTION MARKUP LANGUAGE (SAML)	1
1. INTRODUCTION	10
1.1. NOTATION	10
1.2. SCHEMA ORGANIZATION AND NAMESPACES	10
<i>1.2.1. String and URI Values</i>	<i>11</i>
<i>1.2.2. Time Values.</i>	<i>11</i>
<i>1.2.3. Comparing SAML values</i>	<i>11</i>
1.3. SAML CONCEPTS (NON-NORMATIVE)	11
<i>1.3.1. Overview</i>	<i>12</i>
<i>1.3.2. SAML and URI-Based Identifiers</i>	<i>13</i>
<i>1.3.3. SAML and Extensibility</i>	<i>13</i>
2. SAML ASSERTIONS	14
2.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS	14
2.2. SIMPLE TYPES	14
<i>2.2.1. Simple Types IDType and IDReferenceType</i>	<i>14</i>
<i>2.2.2. Simple Type DecisionType</i>	<i>15</i>
2.3. ASSERTIONS	15
<i>2.3.1. Element <AssertionID></i>	<i>15</i>
<i>2.3.2. Element <Assertion></i>	<i>15</i>
<i>2.3.2.1. Element <Conditions></i>	<i>17</i>
<i>2.3.2.1.1 Attributes NotBefore and NotOnOrAfter</i>	<i>18</i>
<i>2.3.2.1.2 Element <Condition></i>	<i>18</i>
<i>2.3.2.1.3 Elements <AudienceRestrictionCondition> and <Audience></i>	<i>18</i>
<i>2.3.2.2. Elements <Advice> and <AdviceElement></i>	<i>19</i>
2.4. STATEMENTS	19
<i>2.4.1. Element <Statement></i>	<i>19</i>

43	<u>2.4.2. Element <SubjectStatement></u>	<u>19</u>
44	<u>2.4.2.1. Element <Subject></u>	<u>20</u>
45	<u>2.4.2.2. Element <NameIdentifier></u>	<u>20</u>
46	<u>2.4.2.3. Elements <SubjectConfirmation>, <ConfirmationMethod>, and <SubjectConfirmationData></u>	<u>21</u>
47	<u>2.4.3. Element <AuthenticationStatement></u>	<u>22</u>
48	<u>2.4.3.1. Element <SubjectLocality></u>	<u>22</u>
49	<u>2.4.3.2. Element <AuthorityBinding></u>	<u>23</u>
50	<u>2.4.4. Element <AuthorizationDecisionStatement></u>	<u>23</u>
51	<u>2.4.4.1. Element <Action></u>	<u>25</u>
52	<u>2.4.4.2. Element <Evidence></u>	<u>25</u>
53	<u>2.4.5. Element <AttributeStatement></u>	<u>25</u>
54	<u>2.4.5.1. Elements <AttributeDesignator> and <Attribute></u>	<u>26</u>
55	<u>2.4.5.1.1 Element <AttributeValue></u>	<u>26</u>
56	<u>3. SAML PROTOCOL</u>	<u>28</u>
57	<u>3.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS</u>	<u>28</u>
58	<u>3.2. REQUEST S</u>	<u>28</u>
59	<u>3.2.1. Complex Type RequestAbstractType</u>	<u>28</u>
60	<u>3.2.1.1. Element <RespondWith></u>	<u>29</u>
61	<u>3.2.2. Element <Request></u>	<u>29</u>
62	<u>3.2.3. Element <AssertionArtifact></u>	<u>30</u>
63	<u>3.3. QUERIES</u>	<u>30</u>
64	<u>3.3.1. Element <Query></u>	<u>30</u>
65	<u>3.3.2. Element <SubjectQuery></u>	<u>31</u>
66	<u>3.3.3. Element <AuthenticationQuery></u>	<u>31</u>
67	<u>3.3.4. Element <AttributeQuery></u>	<u>32</u>
68	<u>3.3.5. Element <AuthorizationDecisionQuery></u>	<u>32</u>
69	<u>3.4. RESPONSES</u>	<u>33</u>
70	<u>3.4.1. Complex Type ResponseAbstractType</u>	<u>33</u>
71	<u>3.4.2. Element <Response></u>	<u>34</u>
72	<u>3.4.3. Element <Status></u>	<u>34</u>

73	3.4.3.1. Element <StatusCode>	35
74	3.4.3.2. Element <StatusMessage>	36
75	3.4.3.3. Element <StatusDetail>	36
76	3.4.4. Responses to <AuthenticationQuery> and <AttributeQuery>	36
77	4. SAML VERSIONING	38
78	4.1. ASSERTION VERSION	38
79	4.2. REQUEST VERSION	38
80	4.3. RESPONSE VERSION	39
81	5. SAML & XML-SIGNATURE SYNTAX AND PROCESSING	40
82	5.1. SIGNING ASSERTIONS	40
83	5.2. REQUEST /RESPONSE SIGNING	41
84	5.3. SIGNATURE INHERITANCE	41
85	5.3.1. Rationale	41
86	5.3.2. Rules for SAML Signature Inheritance	41
87	5.4. XML SIGNATURE PROFILE	41
88	5.4.1. Signing formats	41
89	5.4.2. CanonicalizationMethod	41
90	5.4.3. Transforms	42
91	5.4.4. KeyInfo	42
92	5.4.5. Binding between statements in a multi-statement assertion	42
93	6. SAML EXTENSIONS	43
94	6.1. ASSERTION SCHEMA EXTENSION	43
95	6.2. PROTOCOL SCHEMA EXTENSION	43
96	6.3. USE OF TYPE DERIVATION AND SUBSTITUTION GROUPS	44
97	7. SAML-DEFINED IDENTIFIERS	45

98	7.1. AUTHENTICATION METHOD IDENTIFIERS	45
99	<i>7.1.1. Password :</i>	<i>45</i>
100	<i>7.1.2. Kerberos</i>	<i>45</i>
101	<i>7.1.3. Secure Remote Password (SRP)</i>	<i>45</i>
102	<i>7.1.4. SSL/TLS Certificate Based Client Authentication:</i>	<i>46</i>
103	<i>7.1.5. X.509 Public Key</i>	<i>46</i>
104	<i>7.1.6. PGP Public Key</i>	<i>46</i>
105	<i>7.1.7. SPKI Public Key</i>	<i>46</i>
106	<i>7.1.8. XKMS Public Key</i>	<i>46</i>
107	<i>7.1.9. XML Digital Signature</i>	<i>46</i>
108	7.2. ACTION NAMESPACE IDENTIFIERS	46
109	<i>7.2.1. Read/Write/Execute/Delete/Control:</i>	<i>47</i>
110	<i>7.2.2. Read/Write/Execute/Delete/Control with Negation:</i>	<i>47</i>
111	<i>7.2.3. Get/Head/Put/Post:</i>	<i>47</i>
112	<i>7.2.4. UNIX File Permissions:</i>	<i>47</i>
113	8. SAML SCHEMA LISTINGS	49
114	<i>8.1. ASSERTION SCHEMA</i>	<i>49</i>
115	<i>8.2. PROTOCOL SCHEMA</i>	<i>52</i>
116	9. REFERENCES	55
117	10. ACKNOWLEDGEMENTS	57
118	APPENDIX A. NOTICES	59
119	ASSERTIONS AND PROTOCOL FOR THE OASIS SECURITY ASSERTION MARKUP	
120	LANGUAGE (SAML)	1
121	1. INTRODUCTION	10
122	<i>1.1. NOTATION</i>	<i>10</i>

123	1.2. SCHEMA ORGANIZATION AND NAMESPACES	10
124	1.2.1. String and URI Values	11
125	1.2.2. Time Values.	11
126	1.2.3. Comparing SAML values	11
127	1.3. SAML CONCEPTS (NON-NORMATIVE)	11
128	1.3.1. Overview	12
129	1.3.2. SAML and URI Based Identifiers	13
130	1.3.3. SAML and Extensibility	13
131	2. SAML ASSERTIONS	14
132	2.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS	14
133	2.2. SIMPLE TYPES	14
134	2.2.1. Simple Types IDType and IDReferenceType	14
135	2.2.2. Simple Type DecisionType	15
136	2.3. ASSERTIONS	15
137	2.3.1. Element <AssertionID>	15
138	2.3.2. Element <Assertion>	15
139	2.3.2.1. Element <Conditions>	17
140	2.3.2.1.1. Attributes NotBefore and NotOnOrAfter	18
141	2.3.2.1.2. Element <Condition>	18
142	2.3.2.1.3. Elements <AudienceRestrictionCondition> and <Audience>	18
143	2.3.2.2. Elements <Advice> and <AdviceElement>	19
144	2.4. STATEMENTS	19
145	2.4.1. Element <Statement>	19
146	2.4.2. Element <SubjectStatement>	19
147	2.4.2.1. Element <Subject>	20
148	2.4.2.2. Element <NameIdentifier>	20
149	2.4.2.3. Elements <SubjectConfirmation>, <ConfirmationMethod>, and <SubjectConfirmationData>	21
150	2.4.3. Element <AuthenticationStatement>	22
151	2.4.3.1. Element <SubjectLocality>	22
152	2.4.3.2. Element <AuthorityBinding>	23

153	2.4.4. Element <AuthorizationDecisionStatement>	23
154	2.4.4.1. Element <Action>	25
155	2.4.4.2. Element <Evidence>	25
156	2.4.5. Element <AttributeStatement>	25
157	2.4.5.1. Elements <AttributeDesignator> and <Attribute>	26
158	2.4.5.1.1. Element <AttributeValue>	26
159	3. SAML PROTOCOL	28
160	3.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS	28
161	3.2. REQUESTS	28
162	3.2.1. Complex Type RequestAbstractType	28
163	3.2.1.1. Element <RespondWith>	29
164	3.2.2. Element <Request>	29
165	3.2.3. Element <AssertionArtifact>	30
166	3.3. QUERIES	30
167	3.3.1. Element <Query>	30
168	3.3.2. Element <SubjectQuery>	31
169	3.3.3. Element <AuthenticationQuery>	31
170	3.3.4. Element <AttributeQuery>	32
171	3.3.5. Element <AuthorizationDecisionQuery>	32
172	3.4. RESPONSES	33
173	3.4.1. Complex Type ResponseAbstractType	33
174	3.4.2. Element <Response>	34
175	3.4.3. Element <Status>	34
176	3.4.3.1. Element <StatusCode>	35
177	3.4.3.2. Element <StatusMessage>	36
178	3.4.3.3. Element <StatusDetail>	36
179	3.4.4. Responses to <AuthenticationQuery> and <AttributeQuery>	36
180	4. SAML VERSIONING	38
181	4.1. ASSERTION VERSION	38

182	4.2. REQUEST VERSION	38
183	4.3. RESPONSE VERSION	39
184	5. SAML & XML SIGNATURE SYNTAX AND PROCESSING	40
185	5.1. SIGNING ASSERTIONS	40
186	5.2. REQUEST /RESPONSE SIGNING	41
187	5.3. SIGNATURE INHERITANCE	41
188	5.3.1. Rationale	41
189	5.3.2. Rules for SAML Signature Inheritance	41
190	5.4. XML SIGNATURE PROFILE	41
191	5.4.1. Signing formats	41
192	5.4.2. CanonicalizationMethod	41
193	5.4.3. Transforms	42
194	5.4.4. KeyInfo	42
195	5.4.5. Binding between statements in a multi-statement assertion	42
196	6. SAML EXTENSIONS	43
197	6.1. ASSERTION SCHEMA EXTENSION	43
198	6.2. PROTOCOL SCHEMA EXTENSION	43
199	6.3. USE OF TYPE DERIVATION AND SUBSTITUTION GROUPS	44
200	7. SAML-DEFINED IDENTIFIERS	45
201	7.1. AUTHENTICATION METHOD IDENTIFIERS	45
202	7.1.1. Password:	46
203	7.1.2. Kerberos	46
204	7.1.3. Secure Remote Password (SRP)	47
205	7.1.4. SSL/TLS Certificate Based Client Authentication:	47
206	7.1.5. X.509 Public Key	47

207	7.1.6. PGP Public Key	47
208	7.1.7. SPKI Public Key	47
209	7.1.8. XKMS Public Key	47
210	7.1.9. XML Digital Signature	48
211	7.2. ACTION NAMESPACE IDENTIFIERS	48
212	7.2.1. Read/Write/Execute/Delete/Control:	48
213	7.2.2. Read/Write/Execute/Delete/Control with Negation:	49
214	7.2.3. Get/Head/Put/Post:	49
215	7.2.4. UNIX File Permissions:	49
216	8. SAML SCHEMA LISTINGS	51
217	8.1. ASSERTION SCHEMA	51
218	8.2. PROTOCOL SCHEMA	54
219	9. REFERENCES	57
220	10. ACKNOWLEDGEMENTS	59
221	APPENDIX A. NOTICES	61
222		

223

1. Introduction

224
225
226
227
228
229

This specification defines the syntax and semantics for XML-encoded SAML assertions, protocol requests, and protocol responses. These constructs are typically embedded in other structures for transport, such as HTTP form POSTs and XML-encoded SOAP messages. The SAML specification for bindings and profiles [SAMLBind] provides frameworks for this embedding and transport. Files containing just the SAML assertion schema [SAML-XSD] and protocol schema [SAMPL-XSD] are available.

230

The following sections describe how to understand the rest of this specification.

231

1.1. Notation

232
233
234

This specification uses schema documents conforming to W3C XML Schema [Schema1] and normative text to describe the syntax and semantics of XML-encoded SAML assertions and protocol messages.

235
236
237

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

238
239

"they MUST only be used where it is actually required for interoperability or to limit behavior which has potential for causing harm (e.g., limiting retransmissions)"

240
241
242
243

These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

244
245

Listings of SAML schemas appear like this.

246

Example code listings appear like this.

247
248
249

Conventional XML namespace prefixes are used throughout the listings in this specification to stand for their respective namespaces (see Section 1.2) as follows, whether or not a namespace declaration is present in the example:

250

?? The prefix `saml`: stands for the SAML assertion namespace.

251

?? The prefix `samlp`: stands for the SAML request-response protocol namespace.

252

?? The prefix `ds`: stands for the W3C XML Signature namespace.

253
254

?? The prefix `xsd`: stands for the W3C XML Schema namespace in example listings. In schema listings, this is the default namespace and no prefix is shown.

255
256

This specification uses the following typographical conventions in text: `<SAMLElement>`, `<ns:ForeignElement>`, `Attribute`, **Datatype**, `OtherCode`.

257

1.2. Schema Organization and Namespaces

258
259

The SAML assertion structures are defined in a schema [SAML-XSD] associated with the following XML namespace:

260

`urn:oasis:names:tc:SAML:1.0:assertion`

261
262

The SAML request-response protocol structures are defined in a schema [SAMPL-XSD] associated with the following XML namespace:

263

`urn:oasis:names:tc:SAML:1.0:protocol`

264 **Note:** The SAML namespace names are temporary and will change when
265 SAML 1.0 is finalized.

266 The assertion schema is imported into the protocol schema. Also imported into both schemas is the
267 schema for XML Signature [**XMLSig-XSD**], which is associated with the following XML namespace:

268 <http://www.w3.org/2000/09/xmldsig#>

269 **1.2.1. String and URI Values**

270 All SAML string and URI values have the types string and anyURI respectively, which are built in to
271 the W3C XML Schema Datatypes specification. All strings in SAML messages **MUST** consist of at
272 least one non-whitespace character (whitespace is defined in [XML 1.0 Sec. 2.3]). Empty and
273 whitespace-only values are disallowed. Also, unless otherwise indicated in this specification, all URI
274 values **MUST** consist of at least one non-whitespace character.

275 **1.2.2. Time Values.**

276 All SAML time values have the type **dateTime**, which is built in to the W3C XML Schema Datatypes
277 specification [**Schema2**] and **MUST** be expressed in UTC form.

278 SAML Requestors and Responders **SHOULD NOT** rely on other applications supporting time
279 resolution finer than milliseconds. Implementations **MUST NOT** generate time instants that specify
280 leap seconds.

281 **1.2.3. Comparing SAML values**

282 Unless otherwise noted, all elements in SAML documents that have the XML Schema "string" type,
283 or a type derived from that, **MUST** be compared using an exact binary comparison. In particular,
284 SAML implementations and deployments **MUST NOT** depend on case-insensitive string
285 comparisons, normalization or trimming of white space, or conversion of locale-specific formats
286 such as numbers or currency. This requirement is intended to conform to the W3C Requirements
287 for String Identity, Matching, and String Indexing [**W3C-CHAR**].

288 If an implementation is comparing values that are represented using different character encodings,
289 the implementation **MUST** use a comparison method that returns the same result as converting
290 both values to the Unicode character encoding (<http://www.unicode.org>), Normalization Form C
291 [**UNICODE-C**] and then performing an exact binary comparison. This requirement is intended to
292 conform to the W3C Character Model for the World Wide Web ([**W3C-CharMod**]), and in particular
293 the rules for Unicode-normalized Text.

294 Applications that compare data received in SAML documents to data from external sources **MUST**
295 take into account the normalization rules specified for XML. Text contained within elements is
296 normalized so that line endings are represented using linefeed characters (ASCII code 10_{Decimal}), as
297 described in section 2.11 of the XML Recommendation [**XML**]. Attribute values defined as strings
298 (or types derived from strings) are normalized as described in section 3.3.3 [**XML**]. All white space
299 characters are replaced with blanks (ASCII code 32_{Decimal}).

300 The SAML specification does not define collation or sorting order for attribute or element values.
301 SAML implementations **MUST NOT** depend on specific sorting orders for values, because these
302 may differ depending on the locale settings of the hosts involved.

303 **1.3. SAML Concepts (Non-Normative)**

304 This section is informative only and is superseded by any contradicting information in the normative
305 text in Section 2 and following. A glossary of SAML terms and concepts [**SAMLGloss**] is available.

306

1.3.1. Overview

307
308
309
310
311

The Security Assertion Markup Language (SAML) is an XML-based framework for exchanging security information. This security information is expressed in the form of assertions about subjects, where a subject is an entity (either human or computer) that has an identity in some security domain. A typical example of a subject is a person, identified by his or her email address in a particular Internet DNS domain.

312
313
314
315
316
317

Assertions can convey information about authentication acts performed by subjects, attributes of subjects, and authorization decisions about whether subjects are allowed to access certain resources. Assertions are represented as XML constructs and have a nested structure, whereby a single assertion might contain several different internal statements about authentication, authorization, and attributes. Note that assertions containing authentication statements merely describe acts of authentication that happened previously.

318
319
320
321
322

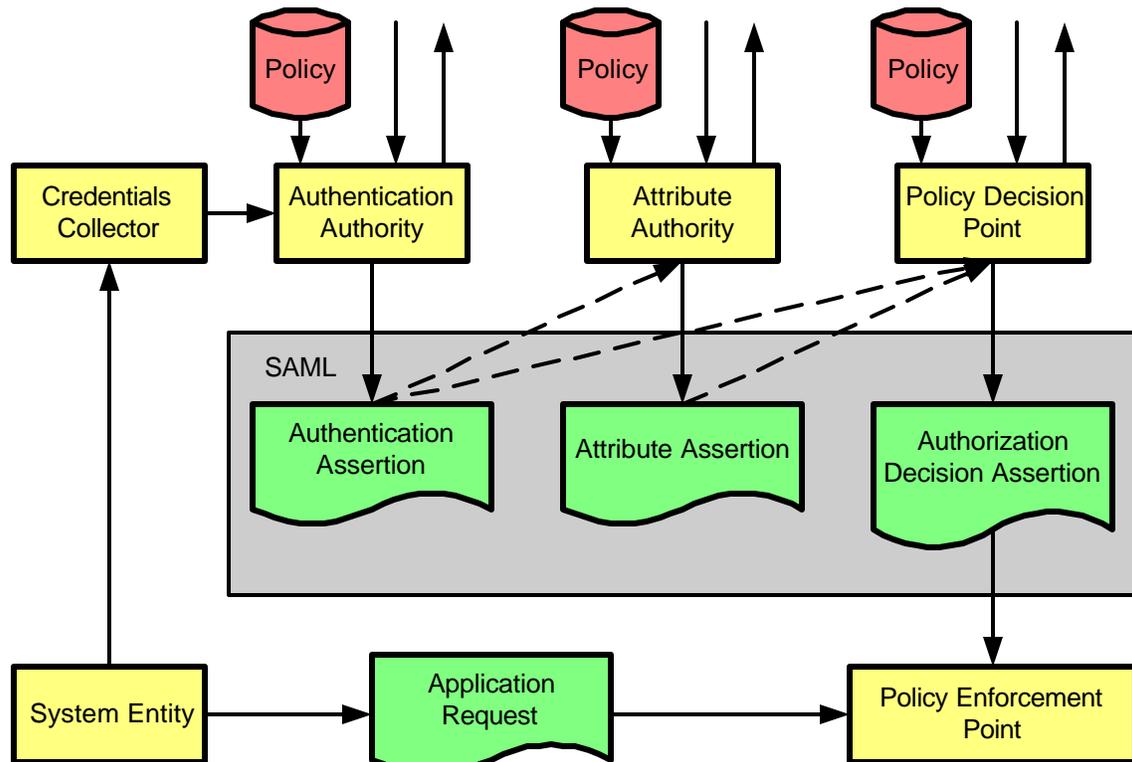
Assertions are issued by SAML authorities, namely, authentication authorities, attribute authorities, and policy decision points. SAML defines a protocol by which clients can request assertions from SAML authorities and get a response from them. This protocol, consisting of XML-based request and response message formats, can be bound to many different underlying communications and transport protocols; SAML currently defines one binding, to SOAP over HTTP.

323
324
325

SAML authorities can use various sources of information, such as external policy stores and assertions that were received as input in requests, in creating their responses. Thus, while clients always consume assertions, SAML authorities can be both producers and consumers of assertions.

326
327

The following model is conceptual only; for example, it does not account for real-world information flow or the possibility of combining of authorities into a single system.



328

329

Figure 1 The SAML Domain Model

330
331

One major design goal for SAML is Single Sign-On (SSO), the ability of a user to authenticate in one domain and use resources in other domains without re-authenticating. However, SAML can be

332 used in various configurations to support additional scenarios as well. Several profiles of SAML are
333 currently being defined that support different styles of SSO and the securing of SOAP payloads.

334 The assertion and protocol data formats are defined in this specification. The bindings and profiles
335 are defined in a separate specification [**SAMLBind**]. A conformance program for SAML is defined
336 in the conformance specification [**SAMLConform**]. Security issues are discussed in a separate
337 security and privacy considerations specification [**SAMLSecure**].

338 **1.3.2. SAML and URI-Based Identifiers**

339 SAML defines some identifiers to manage references to well-known concepts and sets of values.
340 For example, the SAML-defined identifier for the password authentication method is as follows:

341 **urn:oasis:names:tc:SAML:1.0:am:password**

342

343 For another example, the SAML-defined identifier for the set of possible actions on a resource
344 consisting of Read/Write/Execute/Delete/Control is as follows:

345 **urn:oasis:names:tc:SAML:1.0:action:rwdc**

346 These identifiers are defined as Uniform Resource Identifiers (URIs), but they are not necessarily
347 able to be resolved to some Web resource. At times SAML authorities need to use identifier strings
348 of their own design, for example, for assertion IDs or additional kinds of authentication methods not
349 covered by SAML-defined identifiers. In these cases, using a URI form is not required; if it is used, it
350 is not required to be resolvable to some Web resource. However, using URIs – particularly URLs
351 based on the `http:` scheme – is likely to mitigate problems with clashing identifiers to some
352 extent.

353 The Read/Write/Execute/Delete/Control identifier above is an example of a namespace (not in the
354 sense of an XML namespace). SAML uses this namespace mechanism to manage the universe of
355 possible types of actions and possible names of attributes.

356 See section 7 for a list of SAML-defined identifiers.

357 **1.3.3. SAML and Extensibility**

358 The XML formats for SAML assertions and protocol messages have been designed to be
359 extensible.

360 However, it is possible that the use of extensions will harm interoperability and therefore the use of
361 extensions SHOULD be carefully considered.

362 2. SAML Assertions

363 An assertion is a package of information that supplies one or more statements made by an issuer.
364 SAML allows issuers to make three different kinds of assertion statement:

365 ?? **Authentication:** The specified subject was authenticated by a particular means at a
366 particular time.

367 ?? **Authorization Decision:** A request to allow the specified subject to access the specified
368 resource has been granted or denied.

369 ?? **Attribute:** The specified subject is associated with the supplied attributes.

370 Assertions have a nested structure. A series of inner elements representing authentication
371 statements, authorization decision statements, and attribute statements contain the specifics, while
372 an outer generic assertion element provides information that is common to all of the statements.

373 2.1. Schema Header and Namespace Declarations

374 The following schema fragment defines the XML namespaces and other header information for the
375 assertion schema:

```
376 <schema  
377   targetNamespace="urn:oasis:names:tc:SAML:1.0:assertion"  
378   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
379   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"  
380   xmlns="http://www.w3.org/2001/XMLSchema"  
381   elementFormDefault="unqualified">  
382   <import namespace="http://www.w3.org/2000/09/xmldsig#"  
383     schemaLocation="xmldsig-core-schema.xsd"/>  
384   <annotation>  
385     <documentation>draft-sstc-schema-assertion-3231.xsd</documentation>  
386   </annotation>  
387   ...  
388 </schema>
```

389 2.2. Simple Types

390 The following sections define the SAML assertion-related simple types.

391 2.2.1. Simple Types IDType and IDReferenceType

392 The **IDType** simple type is used to declare identifiers to assertions, requests, and responses. The
393 **IDReferenceType** is used to reference identifiers of type **IDType**.

394 Values declared to be of type **IDType** MUST satisfy the following properties:

395 ?? Any party that assigns an identifier MUST ensure that there is negligible probability that that
396 party or any other party will accidentally assign the same identifier to a different data object.

397 ?? Where a data object declares that it has a particular identifier, there MUST be exactly one
398 such declaration.

399 The mechanism by which the SAML Requestor or Responder ensures that the identifier is unique is
400 left to the implementation. In the case that a pseudorandom technique is employed, the probability
401 of two randomly chosen identifiers being identical MUST be less than 2^{-128} and SHOULD be less
402 than 2^{-160} . This requirement MAY be met by applying Base64 encoding to a randomly chosen value
403 128 or 160 bits in length.

404 It is OPTIONAL for an identifier based on **IDType** to be resolvable in principle to some resource. In
405 the case that the identifier is resolvable in principle (for example, the identifier is in the form of a
406 URI reference), it is OPTIONAL for the identifier to be dereferenceable.

407 The following schema fragment defines the **IDType** and **IDReferenceType** simple types:

```
408 <simpleType name="IDType">  
409   <restriction base="string"/>  
410 </simpleType>  
411 <simpleType name="IDReferenceType">  
412   <restriction base="string"/>  
413 </simpleType>
```

414 2.2.2. Simple Type DecisionType

415 The **DecisionType** simple type defines the possible values to be reported as the status of an
416 authorization decision statement.

417 Permit

418 The specified action is permitted.

419 Deny

420 The specified action is denied.

421 IndeterminateThe issuer cannot determine whether the specified action is permitted or denied.

422 The Indeterminate Decision value is used in situations where the issuer requires the ability to
423 provide an affirmative statement that it is not able to issue a decision. Additional information as to
424 the reason for the refusal or inability to provide a decision MAY be returned as <StatusDetail>
425 elements

426

427 The following schema fragment defines the **DecisionType** simple type:

```
428 <simpleType name="DecisionType">  
429   <restriction base="string">  
430     <enumeration value="Permit"/>  
431     <enumeration value="Deny"/>  
432     <enumeration value="Indeterminate"/>  
433   </restriction>  
434 </simpleType>
```

435 2.3. Assertions

436 The following sections define the SAML constructs that contain assertion information.

437 2.3.1. Element <AssertionIDReference>

438 The <AssertionIDReference> element makes a reference to a SAML assertion ~~by means of~~
439 ~~the value of the assertion's AssertionID attribute.~~

440 The following schema fragment defines the <AssertionIDReference> element:

```
441 <element name="AssertionIDReference" type="saml:IDReferenceType"/>
```

442 2.3.2. Element <Assertion>

443 The <Assertion> element is of **AssertionType** complex type. This type specifies the basic
444 information that is common to all assertions, including the following elements and attributes:

445 MajorVersion [Required]

446 The major version of this assertion. The identifier for the version of SAML defined in this
447 specification is 1. Processing of this attribute is specified in Section 3.4.4.

448 MinorVersion [Required]
 449 The minor version of this assertion. The identifier for the version of SAML defined in this
 450 specification is 0. Processing of this attribute is specified in Section 3.4.4.

451 AssertionID [Required]
 452 The identifier for this assertion. It is of type **IDType**, and MUST follow the requirements
 453 specified by that type for identifier uniqueness.

454 Issuer [Required]
 455 The issuer of the assertion. The name of the issuer is provided as a string. The issuer
 456 name SHOULD be unambiguous to the intended relying parties. SAML authorities may use
 457 an identifier such as a URI reference that is designed to be unambiguous regardless of
 458 context.

459 IssueInstant [Required]
 460 The time instant of issue in UTC as described in section 1.2.1.

461 <Conditions> [Optional]
 462 Conditions that MUST be taken into account in assessing the validity of the assertion.

463 <Advice> [Optional]
 464 Additional information related to the assertion that assists processing in certain situations
 465 but which MAY be ignored by applications that do not support its use.

466 <Signature> [Optional]
 467 An XML Signature that authenticates the assertion, see section 5.

468 One or more of the following statement elements:

469 <Statement>
 470 A statement defined in an extension schema.

471 <SubjectStatement>
 472 A subject statement defined in an extension schema.

473 <AuthenticationStatement>
 474 An authentication statement.

475 <AuthorizationDecisionStatement>
 476 An authorization decision statement.

477 <AttributeStatement>
 478 An attribute statement.

479 The following schema fragment defines the <Assertion> element and its **AssertionType**
 480 complex type:

```

481 <element name="Assertion" type="saml:AssertionType"/>
482 <complexType name="AssertionType">
483   <sequence>
484     <element ref="saml:Conditions" minOccurs="0"/>
485     <element ref="saml:Advice" minOccurs="0"/>
486     <choice maxOccurs="unbounded">
487       <element ref="saml:Statement"/>
488       <element ref="saml:SubjectStatement"/>
489       <element ref="saml:AuthenticationStatement"/>
490       <element ref="saml:AuthorizationDecisionStatement"/>
491       <element ref="saml:AttributeStatement"/>
492     </choice>
493     <element ref="ds:Signature" minOccurs="0"/>
494   </sequence>
495   <attribute name="MajorVersion" type="integer" use="required"/>
496   <attribute name="MinorVersion" type="integer" use="required"/>
497   <attribute name="AssertionID" type="saml:IDType" use="required"/>

```

```
498     <attribute name="Issuer" type="string" use="required"/>
499     <attribute name="IssueInstant" type="dateTime" use="required"/>
500 </complexType>
```

501 2.3.2.1. Element <Conditions>

502

503 The <Conditions> element MAY contain the following elements and attributes:

504 NotBefore [Optional]

505 Specifies the earliest time instant at which the assertion is valid. The time value is encoded
506 in UTC as described in section 1.2.1.

507 NotOnOrAfter [Optional]

508 Specifies the time instant at which the assertion has expired. The time value is encoded in
509 UTC as described in section 1.2.1.

510 <Condition> [Any Number]

511 Provides an extension point allowing extension schemas to define new conditions.

512 <AudienceRestrictionCondition> [Any Number]

513 Specifies that the assertion is addressed to a particular audience.

514 The following schema fragment defines the <Conditions> element and its **ConditionsType**
515 complex type:

```
516     <element name="Conditions" type="saml:ConditionsType"/>
517     <complexType name="ConditionsType">
518         <choice minOccurs="0" maxOccurs="unbounded">
519             <element ref="saml:AudienceRestrictionCondition"/>
520             <element ref="saml:Condition"/>
521         </choice>
522         <attribute name="NotBefore" type="dateTime" use="optional"/>
523         <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>
524     </complexType>
```

525 If an assertion contains a <Conditions> element, the validity of the assertion is dependent on the
526 sub-elements and attributes provided. When processing the sub-elements and attributes of a
527 <Conditions> element, the following rules MUST be used in the order shown to determine the
528 overall validity of the assertion:

- 529 1. If no sub-elements or attributes are supplied in the <Conditions> element, then the
530 assertion is considered to be **Valid**.
- 531 2. If any sub-element or attribute of the <Conditions> element is determined to be invalid,
532 then the assertion is **Invalid**.
- 533 3. If any sub-element or attribute of the <Conditions> element cannot be evaluated, then
534 the validity of the assertion cannot be determined and is deemed to be **Indeterminate**.
- 535 4. If all sub-elements and attributes of the <Conditions> element are determined to be
536 **Valid**, then the assertion is considered to be **Valid**.

537 The <Conditions> element MAY be extended to contain additional conditions. If an element
538 contained within a <Conditions> element is encountered that is not understood, the status of the
539 condition cannot be evaluated and the validity status of the assertion MUST be deemed to be
540 **Indeterminate** in accordance with rule 3 above.

541 Note that an assertion that has validity status **Valid** may not be trustworthy by reasons such as not
542 being issued by a trustworthy issuer or not being authenticated by a trustworthy means.

543 **2.3.2.1.1 Attributes *NotBefore* and *NotOnOrAfter***

544 The `NotBefore` and `NotOnOrAfter` attributes specify time limits on the validity of the assertion.

545 The `NotBefore` attribute specifies the time instant at which the validity interval begins. The
546 `NotOnOrAfter` attribute specifies the time instant at which the validity interval has ended.

547 If the value for either `NotBefore` or `NotOnOrAfter` is omitted it is considered unspecified. If the
548 `NotBefore` attribute is unspecified (and if any other conditions that are supplied evaluate to
549 `Valid`), the assertion is valid at any time before the time instant specified by the `NotOnOrAfter`
550 attribute. If the `NotOnOrAfter` attribute is unspecified (and if any other conditions that are supplied
551 evaluate to `Valid`), the assertion is valid from the time instant specified by the `NotBefore`
552 attribute with no expiry. If neither attribute is specified (and if any other conditions that are supplied
553 evaluate to `Valid`), the assertion is valid at any time.

554 The `NotBefore` and `NotOnOrAfter` attributes are defined to have the **dateTime** simple type that
555 is built in to the W3C XML Schema Datatypes specification [**Schema2**]. All time instants are
556 specified in Universal Coordinated Time (UTC) as described in section 1.2.1. Implementations
557 MUST NOT generate time instants that specify leap seconds.

558 **2.3.2.1.2 Element `<Condition>`**

559 The `<Condition>` element serves as an extension point for new conditions. Its
560 **ConditionAbstractType** complex type is abstract; extension elements MUST use the `xsi:type`
561 attribute to indicate the derived type.

562 The following schema fragment defines the `<Condition>` element and its
563 **ConditionAbstractType** complex type:

```
564 <element name="Condition" type="saml:ConditionAbstractType"/>  
565 <complexType name="ConditionAbstractType" abstract="true"/>
```

566 **2.3.2.1.3 Elements `<AudienceRestrictionCondition>` and `<Audience>`**

567 The `<AudienceRestrictionCondition>` element specifies that the assertion is addressed to
568 one or more specific audiences identified by `<Audience>` elements. Although a party that is outside
569 the audiences specified is capable of drawing conclusions from an assertion, the issuer explicitly
570 makes no representation as to accuracy or trustworthiness to such a party. It contains the following
571 elements:

572 `<Audience>`

573 A URI reference that identifies an intended audience. The URI reference MAY identify a
574 document that describes the terms and conditions of audience membership.

575 The `AudienceRestrictionCondition` evaluates to `Valid` if and only if the relying party is a
576 member of one or more of the audiences specified.

577 The issuer of an assertion cannot prevent a party to whom it is disclosed from making a decision on
578 the basis of the information provided. However, the `<AudienceRestrictionCondition>`
579 element allows the issuer to state explicitly that no warranty is provided to such a party in a
580 machine- and human-readable form. While there can be no guarantee that a court would uphold
581 such a warranty exclusion in every circumstance, the probability of upholding the warranty
582 exclusion is considerably improved.

583 The following schema fragment defines the `<AudienceRestrictionCondition>` element and
584 its **AudienceRestrictionConditionType** complex type:

```
585 <element name="AudienceRestrictionCondition"  
586 type="saml:AudienceRestrictionConditionType"/>  
587 <complexType name="AudienceRestrictionConditionType">  
588 <complexContent>  
589 <extension base="saml:ConditionAbstractType">
```

```

590         <sequence>
591             <element ref="saml:Audience" maxOccurs="unbounded" />
592         </sequence>
593     </extension>
594 </complexContent>
595 </complexType>
596 <element name="Audience" type="anyURI" />

```

597 2.3.2.2. Elements <Advice> and <AdviceElement>

598 The <Advice> element contains any additional information that the issuer wishes to provide. This
599 information MAY be ignored by applications without affecting either the semantics or the validity of
600 the assertion.

601 The <Advice> element contains a mixture of zero or more <Assertion> elements,
602 <AssertionIDReference> elements and elements in other namespaces, with lax schema
603 validation in effect for these other elements.

604 Following are some potential uses of the <Advice> element:

- 605 ?? Include evidence supporting the assertion claims to be cited, either directly (through
606 incorporating the claims) or indirectly (by reference to the supporting assertions).
- 607 ?? State a proof of the assertion claims.
- 608 ?? Specify the timing and distribution points for updates to the assertion.

609 The following schema fragment defines the <Advice> element and its **AdviceType** complex type:

```

610 <element name="Advice" type="saml:AdviceType" />
611 <complexType name="AdviceType">
612     <choice minOccurs="0" maxOccurs="unbounded">
613         <element ref="saml:AssertionIDReference" />
614         <element ref="saml:Assertion" />
615         <any namespace="##other" processContents="lax" />
616     </choice>
617 </complexType>

```

618 2.4. Statements

619 The following sections define the SAML constructs that contain statement information.

620 2.4.1. Element <Statement>

621 The <Statement> element is an extension point that allows other assertion-based applications to
622 reuse the SAML assertion framework. Its **StatementAbstractType** complex type is abstract;
623 extension elements MUST use the `xsi:type` attribute to indicate the derived type.

624 The following schema fragment defines the <Statement> element and its
625 **StatementAbstractType** complex type:

```

626 <element name="Statement" type="saml:StatementAbstractType" />
627 <complexType name="StatementAbstractType" abstract="true" />

```

628 2.4.2. Element <SubjectStatement>

629 The <SubjectStatement> element is an extension point that allows other assertion-based
630 applications to reuse the SAML assertion framework. It contains a <Subject> element that allows
631 an issuer to describe a subject. Its **SubjectStatementAbstractType** complex type, which extends
632 **StatementAbstractType**, is abstract; extension elements MUST use the `xsi:type` attribute to
633 indicate the derived type.

634 The following schema fragment defines the <SubjectStatement> element and its
635 **SubjectStatementAbstractType** abstract type:

```
636 <element name="SubjectStatement" type="saml:SubjectStatementAbstractType"/>  
637 <complexType name="SubjectStatementAbstractType" abstract="true">  
638 <complexContent>  
639 <extension base="saml:StatementAbstractType">  
640 <sequence>  
641 <element ref="saml:Subject"/>  
642 </sequence>  
643 </extension>  
644 </complexContent>  
645 </complexType>
```

646 2.4.2.1. Element <Subject>

647 The <Subject> element specifies the principal that is the subject of the statement. It contains
648 either or both of the following elements:

649 <NameIdentifier>

650 An identification of a subject by its name and security domain.

651 <SubjectConfirmation>

652 Information that allows the subject to be authenticated.

653 If the <Subject> element contains both a <NameIdentifier> and a
654 <SubjectConfirmation>, the issuer is asserting that if the relying party performs the specified
655 <SubjectConfirmation>, it can be confident that the entity presenting the assertion to the
656 relying party is the entity that the issuer associates with the <NameIdentifier> A <Subject>
657 element SHOULD NOT identify more than one principal.

658 The following schema fragment defines the <Subject> element and its **SubjectType** complex
659 type:

```
660 <element name="Subject" type="saml:SubjectType"/>  
661 <complexType name="SubjectType">  
662 <choice>  
663 <sequence>  
664 <element ref="saml:NameIdentifier"/>  
665 <element ref="saml:SubjectConfirmation" minOccurs="0"/>  
666 </sequence>  
667 <element ref="saml:SubjectConfirmation"/>  
668 </choice>  
669 </complexType>
```

670 2.4.2.2. Element <NameIdentifier>

671 The <NameIdentifier> element specifies a subject by a combination of a name qualifier, a name
672 and a format. It has the following attributes:

673 NameQualifier [Optional]

674 The security or administrative domain that qualifies the name of the subject.

675 The NameQualifier attribute provides a means to federate names from disparate user
676 stores without collision.

677 Format [Optional]

678 The syntax used to describe the name of the subject

679 The format value MUST be a URI reference. The following URI references are defined by this
680 specification, where only the fragment identifier portion is shown, assuming a base URI of
681 the SAML assertion namespace name.

682 #emailAddress

683 Indicates that the content of the NameIdentifier element is in the form of an email address,

684 specifically "addr-spec" as defined in section 3.4.1 of RFC 2822 [RFC 2822]. An addr-spec
685 has the form local-part@domain. Note that an addr-spec has no phrase (such as a
686 common name) before it, has no comment (text surrounded in parentheses) after it, and is
687 not surrounded by "<" and ">".

688 #X509SubjectName
689 Indicates that the content of the NameIdentifier element is in the form specified for
690 the contents of <ds:X509SubjectName> element in [DSIG]. Implementors should note that
691 [DSIG] specifies encoding rules for X.509 subject names that differ from the rules given in
692 RFC2253 [RFC2253].

693 #WindowsDomainQualifiedName
694 Indicates that the content of the NameIdentifier element is a Windows domain qualified
695 name. A Windows domain qualified user name is a string of the form
696 "DomainName\UserName". The domain name and "\" separator may be omitted.

697 The following schema fragment defines the <NameIdentifier> element and its
698 **NameIdentifierType** complex type:

```
699 <element name="NameIdentifier" type="saml:NameIdentifierType"/>  
700 <complexType name="NameIdentifierType">  
701 <simpleContent>  
702 <extension base="string">  
703 <attribute name="NameQualifier" type="string" use="optional"/>  
704 <attribute name="Format" type="anyURI" use="optional"/>  
705 </extension>  
706 </simpleContent>  
707 </complexType>
```

708 The interpretation of the NameQualifier, and NameIdentifier's content in the case of a Format not
709 specified in this document, are left to individual implementations.

710 Regardless of format, issues of anonymity, pseudonymity, and the persistence of
711 the identifier with respect to the asserting and relying parties, are also
712 implementation-specific.

713 2.4.2.3. Elements <SubjectConfirmation>, <ConfirmationMethod>, and 714 <SubjectConfirmationData>

715 The <SubjectConfirmation> element specifies a subject by supplying data that allows the
716 subject to be authenticated. It contains the following elements in order:

717 <ConfirmationMethod> [One or more]
718 A URI reference that identifies a protocol to be used to authenticate the subject. URI
719 references identifying SAML-defined confirmation methods are currently defined with the
720 SAML profiles in [SAMLBind]. Additional methods may be added by defining new profiles
721 or by private agreement.~~Additional SAML confirmation methods may be defined in future~~
722 ~~OASIS-approved SAML profile specifications.~~

723 <SubjectConfirmationData> [Optional]
724 Additional authentication information to be used by a specific authentication protocol.

725 <ds:KeyInfo> [Optional]
726 An XML Signature [XMLSig] element that specifies a cryptographic key held by the
727 subject.

728 The following schema fragment defines the <SubjectConfirmation> element and its
729 **SubjectConfirmationType** complex type, along with the <SubjectConfirmationData>
730 element and the <ConfirmationMethod> element:

```
731 <element name="SubjectConfirmation" type="saml:SubjectConfirmationType"/>  
732 <complexType name="SubjectConfirmationType">  
733 <sequence>
```

```

734         <element ref="saml:ConfirmationMethod" maxOccurs="unbounded" />
735         <element ref="saml:SubjectConfirmationData" minOccurs="0" />
736         <element ref="ds:KeyInfo" minOccurs="0" />
737     </sequence>
738 </complexType>
739 <element name="SubjectConfirmationData" type="anyType" />
740 <element name="ConfirmationMethod" type="anyURI" />

```

741 2.4.3. Element <AuthenticationStatement>

742 The <AuthenticationStatement> element supplies a statement by the issuer that its subject
743 was authenticated by a particular means at a particular time. It is of type
744 **AuthenticationStatementType**, which extends **SubjectStatementAbstractType** with the addition
745 of the following element and attributes:

746 AuthenticationMethod [Optional]

747 A URI reference that specifies the type of authentication that took place. URI references
748 identifying common authentication protocols are listed in Section 7.

749 AuthenticationInstant [Optional]

750 Specifies the time at which the authentication took place. The time value is encoded in UTC
751 as described in section 1.2.1.

752 <SubjectLocality> [Optional]

753 Specifies the DNS domain name and IP address for the system entity from which the
754 Subject was apparently authenticated.

755 <AuthorityBinding> [Any Number]

756 Indicates that additional information about the subject of the statement may be available.

757 The following schema fragment defines the <AuthenticationStatement> element and its
758 **AuthenticationStatementType** complex type:

```

759 <element name="AuthenticationStatement"
760         type="saml:AuthenticationStatementType" />
761 <complexType name="AuthenticationStatementType">
762     <complexContent>
763         <extension base="saml:SubjectStatementAbstractType">
764             <sequence>
765                 <element ref="saml:SubjectLocality" minOccurs="0" />
766                 <element ref="saml:AuthorityBinding"
767                     minOccurs="0" maxOccurs="unbounded" />
768             </sequence>
769             <attribute name="AuthenticationMethod" type="anyURI" />
770             <attribute name="AuthenticationInstant" type="dateTime" />
771         </extension>
772     </complexContent>
773 </complexType>

```

774 2.4.3.1. Element <SubjectLocality>

775 The <SubjectLocality> element specifies the DNS domain name and IP address for the
776 system entity that was authenticated. It has the following attributes:

777 IPAddress [Optional]

778 The IP address of the system entity that was authenticated.

779 DNSAddress [Optional]

780 The DNS address of the system entity that was authenticated.

781 This element is entirely advisory, since both these fields are quite easily “spoofed” but current
782 practice appears to require its inclusion.

783 The following schema fragment defines the <SubjectLocality> element and its
784 **SubjectLocalityType** complex type:

```
785     <element name="SubjectLocality"  
786           type="saml: SubjectLocalityType" />  
787     <complexType name="SubjectLocalityType">  
788       <attribute name="IPAddress" type="string" use="optional" />  
789       <attribute name="DNSAddress" type="string" use="optional" />  
790     </complexType>
```

791 2.4.3.2. Element <AuthorityBinding>

792 The <AuthorityBinding> element may be used to indicate to a relying party receiving an
793 AuthenticationStatement that a SAML authority may be available to provide additional information
794 about the subject of the statement. A single SAML authority may advertise its presence over
795 multiple protocol bindings, at multiple locations, and as more than one kind of authority by sending
796 multiple elements as needed.

797 *AuthorityKind* [Required]

798 The type of SAML Protocol queries to which the authority described by this element will
799 respond. The value is specified as an XML Schema QName. The acceptable values for
800 *AuthorityKind* are the namespace-qualified names of element types or elements
801 derived from the SAML Protocol Query element (see Section 3.3). For example, an
802 attribute authority would be identified by *AuthorityKind*="samlp:AttributeQuery".
803 For extension schemas, where the actual type of the *samlp:Query* would be identified by
804 an *xsi:type* attribute, the value of *AuthorityKind* MUST be the same as the value of
805 the *xsi:type* attribute for the corresponding query.

806 *Location* [Required]

807 A URI reference describing how to locate and communicate with the authority, the exact
808 syntax of which depends on the protocol binding in use. For example, a binding based on
809 HTTP will be a web URL, while a binding based on SMTP might use the "mailto" scheme.

810 *Binding* [Required]

811 A URI reference identifying the SAML protocol binding to use in communicating with the
812 authority. All SAML protocol bindings will have an assigned URI reference.

813 The following schema fragment defines the <AuthorityBinding> element and its
814 **AuthorityBindingType** complex type and **AuthorityKindType** simple type:

```
815     <element name="AuthorityBinding" type="saml:AuthorityBindingType" />  
816     <complexType name="AuthorityBindingType">  
817       <attribute name="AuthorityKind" type="QName" use="required" />  
818       <attribute name="Location" type="anyURI" use="required" />  
819       <attribute name="Binding" type="anyURI" use="required" />  
820     </complexType>
```

821 2.4.4. Element <AuthorizationDecisionStatement>

822 The <AuthorizationDecisionStatement> element supplies a statement by the issuer that the
823 request for access by the specified subject to the specified resource has resulted in the specified
824 decision on the basis of some optionally specified evidence.

825 The resource is identified by means of a URI reference. In order for the assertion to be interpreted
826 correctly and securely the issuer and relying party MUST interpret each URI reference in a
827 consistent manner. Failure to achieve a consistent URI reference interpretation can result in
828 different authorization decisions depending on the encoding of the resource URI reference. Rules
829 for normalizing URI references are to be found in [RFC 2396]§6

830 *In general, the rules for equivalence and definition of a normal form, if any, are scheme*
831 *dependent. When a scheme uses elements of the common syntax, it will also use the common*
832 *syntax equivalence rules, namely that the scheme and hostname are case insensitive and a*

833 *URL with an explicit ":port", where the port is the default for the scheme, is equivalent to one*
834 *where the port is elided.*

835 To avoid ambiguity resulting from variations in URI encoding SAML requestors and responders
836 SHOULD employ the URI normalized form wherever possible as follows:

837 ?? The assertion issuer SHOULD encode all resource URIs in normalized form.

838 ?? Relying parties SHOULD convert resource URIs to normalized form prior to processing.

839 Inconsistent URI interpretation can also result from differences between the URI syntax and the
840 semantics of an underlying file system. Particular care is required if URIs are employed to specify
841 an access control policy language. The following security conditions should be satisfied by the
842 system which employs SAML assertions:

843 ?? Parts of the URI syntax are case sensitive. If the underlying file system is case insensitive a
844 requestor SHOULD NOT be able to gain access to a denied resource by changing the case
845 of a part of the resource URI.

846 ?? Many file systems support mechanisms such as logical paths and symbolic links which
847 allow users to establish logical equivalences between file system entries. A requestor
848 SHOULD NOT be able to gain access to a denied resource by creating such an
849 equivalence.

850 The <AuthorizationDecisionStatement> element is of type
851 **AuthorizationDecisionStatementType**, which extends **SubjectStatementAbstractType** with the
852 addition of the following elements (in order) and attributes:

853 Resource [Required]

854 A URI reference identifying the resource to which access
855 authorization is sought. It is permitted for this attribute to have
856 the value of the empty URI reference (""), and the meaning is
857 defined to be "the start of the current document", as specified by
858 [RFC 2396]§ 4.2.

859 Decision [Required]

860 The decision rendered by the issuer with respect to the specified resource. The value is of
861 the **DecisionType** simple type.

862 <Action> [One or more]

863 The set of actions authorized to be performed on the specified resource.

864 <Evidence> [Any Number]

865 A set of assertions that the issuer relied on in making the decision.

866 The following schema fragment defines the <AuthorizationDecisionStatement> element
867 and its **AuthorizationDecisionStatementType** complex type:

```
868 <element name="AuthorizationDecisionStatement"  
869 type="saml:AuthorizationDecisionStatementType"/>  
870 <complexType name="AuthorizationDecisionStatementType">  
871 <complexContent>  
872 <extension base="saml:SubjectStatementAbstractType">  
873 <sequence>  
874 <element ref="saml:Action" maxOccurs="unbounded"/>  
875 <element ref="saml:Evidence" minOccurs="0"/>  
876 </sequence>  
877 <attribute name="Resource" type="anyURI" use="required"/>  
878 <attribute name="Decision" type="saml:DecisionType" use="required"/>  
879 </extension>  
880 </complexContent>  
881 </complexType>
```

882 2.4.4.1. Element <Action>

883 The <Action> element specifies an action on the specified resource for which permission is
884 sought. It has the following attribute:

885 Namespace [Optional]

886 A URI reference representing the namespace in which the name of the specified action is
887 to be interpreted. If this element is absent, the namespace
888 urn:oasis:names:tc:SAML:1.0:action:rwdc-negation specified in section 7.2.2 is in effect.

889 *string data* [Required]

890 An action sought to be performed on the specified resource.

891 The following schema fragment defines the <Action> element and its **ActionType** complex type:

```
892 <element name="Action" type="saml:ActionType"/>
893 <complexType name="ActionType">
894   <simpleContent>
895     <extension base="string">
896       <attribute name="Namespace" type="anyURI"/>
897     </extension>
898   </simpleContent>
899 </complexType>
```

900 2.4.4.2. Element <Evidence>

901 The <Evidence> element contains an assertion that the issuer relied on in issuing the
902 authorization decision. It has the **EvidenceType** complex type. It contains one of the following
903 elements:

904 <AssertionIDReference>

905 Specifies an assertion by reference to the value of the assertion's `AssertionID` attribute.

906 <Assertion>

907 Specifies an assertion by value.

908 The provision of an assertion as evidence MAY affect the reliance agreement between the
909 requestor and the Authorization Authority. For example, in the case that the requestor presented an
910 assertion to the Authorization Authority in a request, the Authorization Authority MAY use that
911 assertion as evidence in making its response without endorsing the assertion as valid either to the
912 requestor or any third party.

913 The following schema fragment defines the <Evidence> element and its **EvidenceType** complex
914 type:

```
915 <element name="Evidence" type="saml:EvidenceType"/>
916 <complexType name="EvidenceType">
917   <choice maxOccurs="unbounded">
918     <element ref="saml:AssertionIDReference"/>
919     <element ref="saml:Assertion"/>
920   </choice>
921 </complexType>
```

922 2.4.5. Element <AttributeStatement>

923 The <AttributeStatement> element supplies a statement by the issuer that the specified
924 subject is associated with the specified attributes. It is of type **AttributeStatementType**, which
925 extends **SubjectStatementAbstractType** with the addition of the following element:

926 <Attribute> [One or More]

927 The <Attribute> element specifies an attribute of the subject.

928 The following schema fragment defines the <AttributeStatement> element and its
929 **AttributeStatementType** complex type:

```
930 <element name="AttributeStatement" type="saml:AttributeStatementType" />
931 <complexType name="AttributeStatementType">
932   <complexContent>
933     <extension base="saml:SubjectStatementAbstractType">
934       <sequence>
935         <element ref="saml:Attribute" maxOccurs="unbounded" />
936       </sequence>
937     </extension>
938   </complexContent>
939 </complexType>
```

940 2.4.5.1. Elements <AttributeDesignator> and <Attribute>

941 The <AttributeDesignator> element identifies an attribute name within an attribute
942 namespace. It has the **AttributeDesignatorType** complex type. It is used in an attribute query to
943 request that attribute values within a specific namespace be returned (see 3.3.4 for more
944 information). The <AttributeDesignator> element contains the following XML attributes:

945 AttributeNamespace [Optional]
946 The namespace in which the AttributeName elements are interpreted.

947 AttributeName [Optional]
948 The name of the attribute.

949 The following schema fragment defines the <AttributeDesignator> element and its
950 **AttributeDesignatorType** complex type:

```
951 <element name="AttributeDesignator" type="saml:AttributeDesignatorType" />
952 <complexType name="AttributeDesignatorType">
953   <attribute name="AttributeName" type="string" use="required" />
954   <attribute name="AttributeNamespace" type="anyURI" use="required" />
955 </complexType>
```

956 The <Attribute> element supplies the value for an attribute of an assertion subject. It has the
957 **AttributeType** complex type, which extends **AttributeDesignatorType** with the addition of the
958 following element:

959 <AttributeValue> [Any Number]
960 The value of the attribute.

961 The following schema fragment defines the <Attribute> element and its **AttributeType** complex
962 type:

```
963 <element name="Attribute" type="saml:AttributeType" />
964 <complexType name="AttributeType">
965   <complexContent>
966     <extension base="saml:AttributeDesignatorType">
967       <sequence>
968         <element ref="saml:AttributeValue" maxOccurs="unbounded" />
969       </sequence>
970     </extension>
971   </complexContent>
972 </complexType>
```

973 2.4.5.1.1 Element <AttributeValue>

974 The <AttributeValue> element supplies the value of a specified attribute. It is of the **anyType**
975 simple type, which allows any well-formed XML to appear as the content of the element.

976 If the data content of an AttributeValue element is of a XML Schema simple type (e.g. interger,
977 string, etc) the data type MAY be declared explicitly by means of an `xsi:type` declaration in the

978 <AttributeValue> element. If the attribute value contains structured data the necessary data
979 elements may be defined in an extension schema introduced by means of the `xmlns=` mechanism.

980 The following schema fragment defines the <AttributeValue> element:

981 `<element name="AttributeValue" type="anyType" />`

3. SAML Protocol

SAML assertions MAY be generated and exchanged using a variety of protocols. The bindings and profiles specification for SAML [SAMLBind] describes specific means of transporting assertions using existing widely deployed protocols.

SAML-aware requestors MAY in addition use the SAML request-response protocol defined by the <Request> and <Response> elements. The requestor sends a <Request> element to a SAML authority, and the authority generates a <Response> element, as shown in [Figure 2](#).



Figure 2: SAML Request-Response Protocol

3.1. Schema Header and Namespace Declarations

The following schema fragment defines the XML namespaces and other header information for the protocol schema:

```
<schema
  targetNamespace="urn:oasis:names:tc:SAML:1.0:protocol"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:saml="urn:oasis:names:tc:SAML:1.0:protocol"
  xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  elementFormDefault="unqualified">
  <import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
    schemaLocation="draft-sstc-schema-assertion-3231.xsd"/>
  <import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="xmldsig-core-schema.xsd"/>
  <annotation>
    <documentation>draft-sstc-schema-protocol-3231.xsd</documentation>
  </annotation>
  ...
</schema>
```

3.2. Requests

The following sections define the SAML constructs that contain request information.

3.2.1. Complex Type RequestAbstractType

All SAML requests are of types that are derived from the abstract **RequestAbstractType** complex type. This type defines common attributes and elements that are associated with all SAML requests:

RequestID [Required]

An identifier for the request. It is of type **IDType**, and MUST follow the requirements specified by that type for identifier uniqueness. The values of the **RequestID** attribute in a request and the **InResponseTo** attribute in the corresponding response MUST match.

MajorVersion [Required]

The major version of this request. The identifier for the version of SAML defined in this specification is 1. Processing of this attribute is specified in Section 3.4.2.

1024 MinorVersion [Required]
 1025 The minor version of this request. The identifier for the version of SAML defined in this
 1026 specification is 0. Processing of this attribute is specified in Section 3.4.2.

1027 IssueInstant [Required]
 1028 The time instant of issue of the request. The time value is encoded in UTC as described in
 1029 section 1.2.1.

1030 <RespondWith> [Any Number]
 1031 Each <RespondWith> element specifies a type of response that is acceptable to the
 1032 requestor.

1033 <Signature> [Optional]
 1034 An XML Signature that authenticates the assertion, see section 5.

1035 The following schema fragment defines the **RequestAbstractType** complex type:

```

1036     <complexType name="RequestAbstractType" abstract="true">
1037         <sequence>
1038             <element ref="samlp:RespondWith"
1039                 minOccurs="0" maxOccurs="unbounded"/>
1040             <element ref="ds:Signature" minOccurs="0"/>
1041         </sequence>
1042         <attribute name="RequestID" type="saml:IDType" use="required"/>
1043         <attribute name="MajorVersion" type="integer" use="required"/>
1044         <attribute name="MinorVersion" type="integer" use="required"/>
1045         <attribute name="IssueInstant" type="dateTime" use="required"/>
1046     </complexType>
  
```

1047 3.2.1.1. Element <RespondWith>

1048 The <RespondWith> element specifies the type of Statement the requestor wants from the
 1049 responder. Multiple <RespondWith> elements MAY be included to indicate that the requestor will
 1050 accept assertions containing any of the specified types. If no <RespondWith> element is given,
 1051 the responder may return assertions containing statements of any type.

1052 If the requestor sends one or more <RespondWith> elements, the responder MUST NOT respond
 1053 with assertions containing statements of any type not specified in one of the <RespondWith>
 1054 elements.

1055 NOTE: Inability to find assertions that meet <RespondWith> criteria should be treated identical to
 1056 any other query for which no assertions are available. In both cases a status of success would
 1057 normally be returned in the Response message, but no assertions to be found therein.

1058 <RespondWith> element values are XML QNames. The XML namespace and name specifically
 1059 refer to the namespace and element name of the Statement element, exactly as for the
 1060 saml:AuthorityKind attribute; see section 2.4.3.2. For example, a requestor that wishes to
 1061 receive assertions containing only attribute statements must specify
 1062 <RespondWith>saml:AttributeStatement</RespondWith>. To specify extension types,
 1063 the <RespondWith> element MUST contain exactly the extension element type as specified in the
 1064 xsi:type attribute on the corresponding element.

1065 The following schema fragment defines the <RespondWith> element:

```

1066     <element name="RespondWith" type="QName"/>
  
```

1067 3.2.2. Element <Request>

1068 The <Request> element specifies a SAML request. It provides either a query or a request for a
 1069 specific assertion identified by <AssertionIDReference> or <AssertionArtifact>. It has

1070 the complex type **RequestType**, which extends **RequestAbstractType** by adding a choice of one
1071 of the following elements:

1072 <Query>

1073 An extension point that allows extension schemas to define new types of query.

1074 <SubjectQuery>

1075 An extension point that allows extension schemas to define new types of query that specify
1076 a single SAML subject.

1077 <AuthenticationQuery>

1078 Makes a query for authentication information.

1079 <AttributeQuery>

1080 Makes a query for attribute information.

1081 <AuthorizationDecisionQuery>

1082 Makes a query for an authorization decision.

1083 <AssertionIDReference> [One or more]

1084 Requests assertions by reference to its assertion identifier.

1085 <AssertionArtifact> [One or more]

1086 Requests assertions by supplying an assertion artifact that represents it.

1087 The following schema fragment defines the <Request> element and its **RequestType** complex
1088 type:

```
1089 <element name="Request" type="samlp:RequestType"/>  
1090 <complexType name="RequestType" >  
1091   <complexContent>  
1092     <extension base="samlp:RequestAbstractType">  
1093       <choice>  
1094         <element ref="samlp:Query"/>  
1095         <element ref="samlp:SubjectQuery"/>  
1096         <element ref="samlp:AuthenticationQuery"/>  
1097         <element ref="samlp:AttributeQuery"/>  
1098         <element ref="samlp:AuthorizationDecisionQuery"/>  
1099         <element ref="saml:AssertionIDReference" maxOccurs="unbounded"/>  
1100         <element ref="samlp:AssertionArtifact" maxOccurs="unbounded"/>  
1101       </choice>  
1102     </extension>  
1103   </complexContent>  
1104 </complexType>
```

1105 3.2.3. Element <AssertionArtifact>

1106 The <AssertionArtifact> element is used to specify the assertion artifact that represents an
1107 assertion.

1108 The following schema fragment defines the <AssertionArtifact> element:

```
1109 <element name="AssertionArtifact" type="string"/>
```

1110 3.3. Queries

1111 The following sections define the SAML constructs that contain query information.

1112 3.3.1. Element <Query>

1113 The <Query> element is an extension point that allows new SAML queries to be defined. Its
1114 **QueryAbstractType** is abstract; extension elements **MUST** use the `xsi:type` attribute to indicate

1115 the derived type. **QueryAbstractType** is the base type from which all SAML query elements are
1116 derived.

1117 The following schema fragment defines the <Query> element and its **QueryAbstractType**
1118 complex type:

```
1119 <element name="Query" type="samlp:QueryAbstractType" />  
1120 <complexType name="QueryAbstractType" abstract="true" />
```

1121 3.3.2. Element <SubjectQuery>

1122 The <SubjectQuery> element is an extension point that allows new SAML queries that specify a
1123 single SAML subject. Its **SubjectQueryAbstractType** complex type is abstract; extension elements
1124 MUST use the `xsi:type` attribute to indicate the derived type. **SubjectQueryAbstractType** adds
1125 the <Subject> element.

1126 The following schema fragment defines the <SubjectQuery> element and its
1127 **SubjectQueryAbstractType** complex type:

```
1128 <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType" />  
1129 <complexType name="SubjectQueryAbstractType" abstract="true">  
1130 <complexContent>  
1131 <extension base="samlp:QueryAbstractType">  
1132 <sequence>  
1133 <element ref="saml:Subject" />  
1134 </sequence>  
1135 </extension>  
1136 </complexContent>  
1137 </complexType>
```

1138 3.3.3. Element <AuthenticationQuery>

1139 The <AuthenticationQuery> element is used to make the query “What assertions containing
1140 authentication statements are available for this subject?” A successful response will be in the form
1141 of assertions containing authentication statements.

1142 Note: The <AuthenticationQuery> MAY NOT be used as a request for a new authentication
1143 using credentials provided in the request. The <AuthenticationQuery> is a request for
1144 statements about authentication acts which have occurred in a previous interaction between the
1145 indicated principal and the Authentication Authority.

1146 This element is of type **AuthenticationQueryType**, which extends **SubjectQueryAbstractType**
1147 with the addition of the following element:

1148 <AuthenticationMethod> [Optional]

1149 A filter for possible responses. If it is present, the query made is “What assertions
1150 containing authentication statements do you have for this subject with the supplied
1151 authentication method?”

1152 In response to an authentication query, a responder returns assertions with authentication
1153 statements as follows:

1154 ?? First, rules given in section 3.4.4 for matching against the <Subject> element of the query
1155 identify the assertions that may be returned.

1156 ?? Further, if the <AuthenticationMethod> element is present in the query, at least one
1157 <AuthenticationMethod> element in the set of returned assertions MUST match. It is
1158 OPTIONAL for the complete set of all such matching assertions to be returned in the
1159 response.

1160 The <Subject> element in the returned assertions MUST be identical to the <Subject> element
1161 of the query. If the <ConfirmationMethod> element is present in the query, at least one

1162 <ConfirmationMethod> element in the response MUST match. It is OPTIONAL for the complete
1163 set of all such matching assertions to be returned in the response.

1164 The following schema fragment defines the <AuthenticationQuery> type and its
1165 **AuthenticationQueryType** complex type:

```
1166     <element name="AuthenticationQuery" type="samlp:AuthenticationQueryType"/>  
1167     <complexType name="AuthenticationQueryType">  
1168         <complexContent>  
1169             <extension base="samlp:SubjectQueryAbstractType">  
1170                 <attribute name="AuthenticationMethod" type="anyURI"/>  
1171             </extension>  
1172         </complexContent>  
1173     </complexType>
```

1174 3.3.4. Element <AttributeQuery>

1175 The <AttributeQuery> element is used to make the query “Return the requested attributes for
1176 this subject.” A successful response will be in the form of assertions containing attribute statements.
1177 This element is of type **AttributeQueryType**, which extends **SubjectQueryAbstractType** with the
1178 addition of the following element and attribute:

1179 Resource [Optional]

1180 The Resource attribute if present specifies that the attribute query is made in response to a
1181 specific authorization decision relating to the resource. The responder MAY use the
1182 resource attribute to establish the scope of the request. It is permitted for this attribute to
1183 have the value of the empty URI reference (“”), and the meaning is defined to be “the start
1184 of the current document”, as specified by **[RFC 2396]**§ 4.2.

1185 If the resource attribute is specified and the responder does not wish to support resource-
1186 specific attribute queries, or if the resource value provided is invalid or unrecognized, then it
1187 SHOULD respond with a top-level StatusCode value of Responder and a second-level
1188 code value of ResourceNotRecognized

1189 <AttributeDesignator> [Any Number] (see Section 2.4.5.1)

1190 Each <AttributeDesignator> element specifies an attribute whose value is to be
1191 returned. If no attributes are specified, it indicates that all attributes allowed by policy are
1192 requested.

1193 The following schema fragment defines the <AttributeQuery> element and its
1194 **AttributeQueryType** complex type:

```
1195     <element name="AttributeQuery" type="samlp:AttributeQueryType"/>  
1196     <complexType name="AttributeQueryType">  
1197         <complexContent>  
1198             <extension base="samlp:SubjectQueryAbstractType">  
1199                 <sequence>  
1200                     <element ref="saml:AttributeDesignator"  
1201                         minOccurs="0" maxOccurs="unbounded"/>  
1202                 </sequence>  
1203                 <attribute name="Resource" type="anyURI reference" use="optional"/>  
1204             </extension>  
1205         </complexContent>  
1206     </complexType>
```

1207 3.3.5. Element <AuthorizationDecisionQuery>

1208 The <AuthorizationDecisionQuery> element is used to make the query “Should these
1209 actions on this resource be allowed for this subject, given this evidence?” A successful response
1210 will be in the form of assertions containing authorization decision statements. This element is of
1211 type **AuthorizationDecisionQueryType**, which extends **SubjectQueryAbstractType** with the
1212 addition of the following elements and attribute:

1213 Resource [Required]
1214 A URI reference indicating the resource for which authorization is requested.
1215 <Action> [One or More]
1216 The actions for which authorization is requested.
1217 <Evidence> [Any Number]
1218 An assertion that the responder MAY rely on in making its response.

1219 The following schema fragment defines the <AuthorizationDecisionQuery> element and its
1220 **AuthorizationDecisionQueryType** complex type:

```
1221     <element name="AuthorizationDecisionQuery "  
1222 type="samlp:AuthorizationDecisionQueryType"/>  
1223     <complexType name="AuthorizationDecisionQueryType ">  
1224       <complexContent>  
1225         <extension base="samlp:SubjectQueryAbstractType">  
1226           <sequence>  
1227             <element ref="saml:Action" maxOccurs="unbounded"/>  
1228             <element ref="saml:Evidence"  
1229               minOccurs="0" maxOccurs="unbounded" />  
1230           </sequence>  
1231           <attribute name="Resource" type="anyURI" use="required" />  
1232         </extension>  
1233       </complexContent>  
1234     </complexType>
```

1235 3.4. Responses

1236 The following sections define the SAML constructs that contain response information.

1237 3.4.1. Complex Type ResponseAbstractType

1238 All SAML responses are of types that are derived from the abstract **ResponseAbstractType**
1239 complex type. This type defines common attributes and elements that are associated with all SAML
1240 responses:

1241 ResponseID [Required]
1242 An identifier for the response. It is of type **IDType**, and MUST follow the requirements
1243 specified by that type for identifier uniqueness.

1244 InResponseTo [Optional]
1245 A reference to the identifier of the request to which the response corresponds, if any. If the
1246 response is not generated in response to a request, or if the RequestID of a request cannot
1247 be determined (because the request is malformed), then this attribute MUST NOT be
1248 present. Otherwise, it MUST be present and match the value of the corresponding
1249 RequestID attribute.

1250 MajorVersion [Required]
1251 The major version of this response. The identifier for the version of SAML defined in this
1252 specification is 1. Processing of this attribute is specified in Section 3.4.4.

1253 MinorVersion [Required]
1254 The minor version of this response. The identifier for the version of SAML defined in this
1255 specification is 0. Processing of this attribute is specified in Section 3.4.4.

1256 IssueInstant [Optional]
1257 The time instant of issue of the request. The time value is encoded in UTC as described in
1258 section 1.2.1.

1259 Recipient [Optional]
1260 The intended recipient of this response. This is useful to prevent malicious forwarding of
1261 responses to unintended recipients, a protection that is required by some use profiles. It is
1262 set by the generator of the response to a URI reference that identifies the intended
1263 recipient. If present, the actual recipient MUST check that the URI reference identifies the
1264 recipient or a resource managed by the recipient. If it does not, the response MUST be
1265 discarded.

1266 <Signature> [Optional]
1267 An XML Signature that authenticates the assertion, see section 5.

1268 The following schema fragment defines the **ResponseAbstractType** complex type:

```
1269 <complexType name="ResponseAbstractType" abstract="true" >  
1270 <sequence>  
1271 <element ref = "ds:Signature" minOccurs="0"/>  
1272 </sequence>  
1273 <attribute name="ResponseID" type="saml:IDType" use="required"/>  
1274 <attribute name="InResponseTo" type="saml:IDReferenceType"  
1275 use="optional"/>  
1276 <attribute name="MajorVersion" type="integer" use="required"/>  
1277 <attribute name="MinorVersion" type="integer" use="required"/>  
1278 <attribute name="IssueInstant" type="dateTime" use="required"/>  
1279 <attribute name="Recipient" type="anyURI" use="optional"/>  
1280 </complexType>
```

1281 3.4.2. Element <Response>

1282 The <Response> element specifies the status of the corresponding SAML request and a list of
1283 zero or more assertions that answer the request. It has the complex type **ResponseType**, which
1284 extends **ResponseAbstractType** by adding the following elements (in an unbounded mixture):

1285 <Status> [Required] (see Section 3.4.3)
1286 A code representing the status of the corresponding request.

1287 <Assertion> [Any Number] (see Section 2.3.2)
1288 Specifies an assertion by value.

1289 The following schema fragment defines the <Response> element and its **ResponseType** complex
1290 type:

```
1291 <element name="Response" type="samlp:ResponseType" />  
1292 <complexType name="ResponseType" >  
1293 <complexContent>  
1294 <extension base="samlp:ResponseAbstractType" >  
1295 <sequence>  
1296 <element ref="samlp:Status" />  
1297 <element ref="saml:Assertion"  
1298 minOccurs="0" maxOccurs="unbounded" />  
1299 </sequence>  
1300 </extension>  
1301 </complexContent>  
1302 </complexType>
```

1303 3.4.3. Element <Status>

1304 The <Status> element :

1305 <StatusCode> [Required]
1306 A code representing the status of the corresponding request.

1307 <StatusMessage> [Any Number]
1308 A message which MAY be returned to an operator.

1309 <StatusDetail> [Optional]
1310 Specifies additional information concerning an error condition.

1311 The following schema fragment defines the <Status> element and its **StatusType** complex type:

```
1312 <element name="Status" type="samlp:StatusType"/>  
1313 <complexType name="StatusType">  
1314 <sequence>  
1315 <element ref="samlp:StatusCode"/>  
1316 <element ref="samlp:StatusMessage"  
1317 minOccurs="0" maxOccurs="unbounded"/>  
1318 <element ref="samlp:StatusDetail" minOccurs="0"/>  
1319 </sequence>  
1320 </complexType>
```

1321 3.4.3.1. Element <StatusCode>

1322 The <StatusCode> element specifies one or more nested codes representing the status of the
1323 corresponding request. top-most code value MUST be one of the values defined below.
1324 Subsequent nested code values, if present, may provide more specific information concerning a
1325 particular error.

1326 Value [Required]
1327 The status code value as defined below.

1328 <StatusCode> [Optional]
1329 An optional subordinate status code value that provides more specific information on an
1330 error condition.

1331 The following top-level **StatusCode** Value QNames are defined. The responder MUST NOT
1332 include a code not listed below except by nesting it below one of the listed values.

1333 Success
1334 The request succeeded.

1335 VersionMismatch
1336 The receiver could not process the request because the version was incorrect.

1337 Receiver
1338 The request could not be performed due to an error at the receiving end.

1339 Sender
1340 The request could not be performed due to an error in the sender or in the request

1341 The following second-level status codes are referenced at various places in the specification.
1342 Additional subcodes MAY be defined in future versions of the SAML specification.

1343 RequestVersionTooHigh
1344 The protocol version specified in the request is a major upgrade from the highest protocol
1345 version supported by the responder.

1346 RequestVersionTooLow
1347 The responder cannot respond to the particular request using the SAML version specified
1348 in the request because it is too low.

1349 RequestVersionDeprecated
1350 The responder does not respond to any requests with the protocol version specified in the
1351 request.

1352 TooManyResponses
1353 The response would contain more elements than the responder will return.

1354 RequestDenied
1355 The responder is able to process the request but has chosen not to respond. MAY be used

1356 when the responder is concerned about the security context of the request or the sequence
1357 of requests received from a particular client.

1358 All status code values defined in this document are QNames associated with the SAML protocol
1359 namespace [SAML] and MUST be prefixed appropriately when they appear in SAML messages.
1360 SAML extensions and SAML Responders are free to define more specific status codes in other
1361 namespaces, but MAY NOT define additional codes in either the SAML assertion or protocol
1362 namespaces.

1363 The QNames defined as status codes SHOULD only be used in the StatusCode element's Value
1364 attribute and have the above semantics only in that context.

1365 The following schema fragment defines the <StatusCode> element and its **StatusCodeType**
1366 complex type:

```
1367 <element name="StatusCode" type="samlp:StatusCodeType" />  
1368 <complexType name="StatusCodeType">  
1369 <sequence>  
1370 <element ref="samlp:StatusCode" minOccurs="0" />  
1371 </sequence>  
1372 <attribute name="Value" type="QName" use="required" />  
1373 </complexType>
```

1374 3.4.3.2. Element <StatusMessage>

1375 The <StatusMessage> element specifies a message that MAY be returned to an operator:

1376 The following schema fragment defines the <StatusMessage> element and its
1377 **StatusMessageType** complex type:

```
1378 <element name="StatusMessage" type="string" />
```

1379 3.4.3.3. Element <StatusDetail>

1380 The <StatusDetail> element MAY be used to specify additional information concerning an error
1381 condition.

1382 The following schema fragment defines the <StatusDetail> element and its **StatusDetailType**
1383 complex type:

```
1384 <element name="StatusDetail" type="samlp:StatusDetailType" />  
1385 <complexType name="StatusDetailType">  
1386 <sequence>  
1387 <any namespace="##any"  
1388 processContents="lax" minOccurs="0" maxOccurs="unbounded" />  
1389 </sequence>  
1390 </complexType>
```

1391 3.4.4. Responses to <AuthenticationQuery> and <AttributeQuery>

1392 Responses to Authentication and Attribute queries are constructed by matching against the
1393 <saml:Subject> element found within the <AuthenticationQuery> or <AttributeQuery>
1394 elements. In response to these queries, every assertion returned by a SAML responder MUST
1395 contain at least one statement whose <saml:Subject> element **strongly matches** the
1396 <saml:Subject> element found in the query.

1397 A <saml:Subject> element S1 strongly matches S2 if and only if:

- 1398 1 If S2 includes a <saml:NameIdentifier> element, then S1 must include an identical
1399 <saml:NameIdentifier> element.
- 1400 2 If S2 includes a <saml:SubjectConfirmation> element, then S1 must include an
1401 identical <saml:SubjectConfirmation> element.

1402 If the responder cannot provide an assertion with any statement(s) satisfying the constraints
1403 expressed by a query, the <saml:Response> element MUST NOT contain an <assertion> element
1404 and MUST include a <saml:StatusCode> with value "Success". It MAY return a
1405 <saml:StatusMessage> with additional information.

4. SAML Versioning

1406

1407 SAML version information appears in the following elements:

1408 ?? <Assertion>

1409 ?? <Request>

1410 ?? <Response>

1411 The version numbering of the SAML assertion is independent of the version number of the SAML
1412 request-response protocol. The version information for each consists of a major version number
1413 and a minor version number, both of which are integers. In accordance with industry practice a
1414 version number SHOULD be presented to the user in the form *Major.Minor*. This document defines
1415 SAML Assertions 1.0 and SAML Protocol 1.0.

1416 The version number $Major_B.Minor_B$ is higher than the version number $Major_A.Minor_A$ if and only if:

1417 $Major_B > Major_A ? ((Major_B = Major_A) ? Minor_B > Minor_A)$

1418 Each revision of SAML SHALL assign version numbers to assertions, requests, and responses that
1419 are the same as or higher than the corresponding version number in the SAML version that
1420 immediately preceded it.

1421 New versions of SAML SHALL assign new version numbers as follows:

1422 ?? **Documentation change:** $(Major_B = Major_A) ? (Minor_B > Minor_A)$

1423 If the major and minor version numbers are unchanged, the new version *B* only introduces
1424 changes to the documentation that raise no compatibility issues with an implementation of
1425 version *A*.

1426 ?? **Minor upgrade:** $(Major_B = Major_A) ? (Minor_B > Minor_A)$

1427 If the major version number of versions *A* and *B* are the same and the minor version
1428 number of *B* is higher than that of *A*, the new SAML version MAY introduce changes to the
1429 SAML schema and semantics but any changes that are introduced in *B* SHALL be
1430 compatible with version *A*.

1431 ?? **Major upgrade:** $Major_B > Major_A$

1432 If the major version of *B* number is higher than the major version of *A*, Version *B* MAY
1433 introduce changes to the SAML schema and semantics that are incompatible with *A*.

4.1. Assertion Version

1434

1435 A SAML authority MUST NOT issue any assertion whose version number is not supported.

1436 A SAML relying party MUST reject any assertion whose major version number is not supported.

1437 A SAML relying party MAY reject any assertion whose version number is higher than the highest
1438 supported version.

4.2. Request Version

1439

1440 A SAML authority SHOULD issue requests that specify the highest SAML version supported by
1441 both the sender and recipient.

1442 If the SAML authority does not know the capabilities of the recipient it should assume that it
1443 supports the highest SAML version supported by the sender.

1444 **4.3. Response Version**

1445 A SAML authority **MUST NOT** issue responses that specify a higher SAML version number than the
1446 corresponding request.

1447 A SAML authority **MUST NOT** issue a response that has a major version number that is lower than
1448 the major version number of the corresponding request except to report the error
1449 `RequestVersionTooHigh`.

1450 An error response resulting from incompatible protocol versions **MUST** result in reporting a top-level
1451 `StatusCode` value of `VersionMismatch`, and **MAY** result in reporting one of the following second-
1452 level values:

1453 `RequestVersionTooHigh`

1454 The protocol version specified in the request is a major upgrade from the highest protocol
1455 version supported by the responder.

1456 `RequestVersionTooLow`

1457 The responder cannot respond to the particular request using the SAML version specified
1458 in the request because it is too low.

1459 `RequestVersionDeprecated`

1460 The responder does not respond to any requests with the protocol version specified in the
1461 request.

5. SAML & XML-Signature Syntax and Processing

SAML Assertions, Request and Response messages may be signed, with the following benefits:

?? An Assertion signed by the asserting party (AP). This supports :

- (1) Message integrity
- (2) Authentication of the asserting party to a relying party (RP)
- (3) If the signature is based on the asserting party's public-private key pair, then it also provides for non-repudiation of origin.

?? A SAML request or a SAML response message signed by the message originator. This supports :

- (1) Message integrity
- (2) Authentication of message origin to a destination
- (3) If the signature is based on the originator's public-private key pair, then it also provides for non-repudiation of origin.

Note :

?? SAML documents may be the subject of signatures from different packaging contexts. **[XMLSig]** provides a framework for signing in XML and is the framework of choice. However, signing may also take place in the context of S/MIME or Java objects that contain SAML documents. One goal is to ensure compatibility with this type of "foreign" digital signing.

?? It is useful to characterize situations when a digital signature is NOT required in SAML.

Assertions:

The asserting party has provided the assertion to the relying party, authenticated by means other than digital signature and the channel is secure. In other words, the RP has obtained the assertion from the AP directly (no intermediaries) through a secure channel and the AP has authenticated to the RP.

Request/Response messages:

The originator has authenticated to the destination and the destination has obtained the assertion directly from the originator (no intermediaries) through secure channel(s).

Many different techniques are available for "direct" authentication and secure channel between two parties. The list includes SSL, HMAC, password-based login etc. Also the security requirement depends on the communicating applications and the nature of the assertion transported.

All other contexts require the use of digital signature for assertions and request and response messages. Specifically:

- (1) An assertion obtained by a relying party from an entity other than the asserting party **MUST** be signed by the asserting party.
- (2) A SAML message arriving at a destination from an entity other than the originating site **MUST** be signed by the origin site.

5.1. Signing Assertions

All SAML assertions **MAY** be signed using the XML Signature. This is reflected in the assertion schema – Section 2.3.

1504 **5.2. Request/Response Signing**

1505 All SAML requests and responses MAY be signed using the XML Signature. This is reflected in the
1506 schema – Section 3.2 & 3.4.

1507 **5.3. Signature Inheritance**

1508 **5.3.1. Rationale**

1509 SAML assertions may be embedded within request or response messages or other XML
1510 messages, which may be signed. Request or response messages may themselves be contained
1511 within other messages that are based on other XML messaging frameworks (e.g., SOAP) and the
1512 composite object may be the subject of a signature. Another possibility is that SAML assertions or
1513 request/response messages are embedded within a non-XML messaging object (e.g., MIME
1514 package) and signed.

1515 In such a case, the SAML sub-message (Assertion, request, response) may be viewed as inheriting
1516 a signature from the "super-signature" over the enclosing object, provided certain constraints are
1517 met.

1518 (1) An assertion may be viewed as inheriting a signature from a super signature, if the super
1519 signature applies all the elements within the assertion.

1520 A SAML request or response may be viewed as inheriting a signature from a super signature, if the
1521 super signature applies to all of the elements within the response.

1522 **5.3.2. Rules for SAML Signature Inheritance**

1523 Signature inheritance occurs when SAML message (assertion/request/response) is not signed but
1524 is enclosed within signed SAML such that the signature applies to all of the elements within the
1525 message. In such a case, the SAML message is said to inherit the signature and may be
1526 considered equivalent to the case where it is explicitly signed. The SAML message inherits the
1527 "closest enclosing signature".

1528 But if SAML messages need to be passed around by themselves, or embedded in other messages,
1529 they would need to be signed as per section 5.1

1530 **5.4. XML Signature Profile**

1531 The XML Signature [**XMLSig**] specification calls out a general XML syntax for signing data with
1532 many flexibilities and choices. This section details the constraints on these facilities so that SAML
1533 processors do not have to deal with the full generality of XML Signature processing.

1534 **5.4.1. Signing formats**

1535 XML Signature has three ways of representing signature in a document viz: enveloping, enveloped
1536 and detached.

1537 SAML assertions and protocols MUST use the enveloped signatures for signing assertions and
1538 protocols. SAML processors should support use of RSA signing and verification for public key
1539 operations.

1540 **5.4.2. CanonicalizationMethod**

1541 XML Signature REQUIRES the Canonical XML (omits comments)
1542 (<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>). SAML implementations SHOULD use
1543 Canonical XML with no comments.

1544 **5.4.3. Transforms**

1545 [XMLSig] REQUIRES the enveloped signature transform
1546 <http://www.w3.org/2000/09/xmlsig#enveloped-signature>

1547 **5.4.4. KeyInfo**

1548 SAML does not restrict or impose any restrictions in this area. Therefore following [XMLSig]
1549 keyInfo may be absent.

1550 **5.4.5. Binding between statements in a multi-statement assertion**

1551 Use of signing does not affect semantics of statements within assertions in any way, as stated in
1552 this document Sections 1 through 4.

1553

6. SAML Extensions

1554 The SAML schemas support extensibility. An example of an application that extends SAML
1555 assertions is the XTAML system for management of embedded trust roots [XTAML]. The following
1556 sections explain how to use the extensibility features in SAML to create extension schemas.

1557 Note that elements in the SAML schemas are not blocked from substitution, so that all SAML
1558 elements MAY serve as the head element of a substitution group. Also, types are not defined as
1559 *final*, so that all SAML types MAY be extended and restricted. The following sections discuss
1560 only elements that have been specifically designed to support extensibility.

6.1. Assertion Schema Extension

1562 The SAML assertion schema is designed to permit separate processing of the assertion package
1563 and the statements it contains, if the extension mechanism is used for either part.

1564 The following elements are intended specifically for use as extension points in an extension
1565 schema; their types are set to *abstract*, so that the use of an *xsi:type* attribute with these
1566 elements is REQUIRED:

1567 ?? <Assertion>

1568 ?? <Condition>

1569 ?? <Statement>

1570 ?? <SubjectStatement>

1571 ?? <AdviceElement>

1572 In addition, the following elements that are directly usable as part of SAML MAY be extended:

1573 ?? <AuthenticationStatement>

1574 ?? <AuthorizationDecisionStatement>

1575 ?? <AttributeStatement>

1576 ?? <AudienceRestrictionCondition>

1577 Finally, the following elements are defined to allow elements from arbitrary namespaces within
1578 them, which serves as a built-in extension point without requiring an extension schema:

1579 ?? <AttributeValue>

1580 ?? <Advice>

6.2. Protocol Schema Extension

1582 The following elements are intended specifically for use as extension points in an extension
1583 schema; their types are set to *abstract*, so that the use of an *xsi:type* attribute with these
1584 elements is REQUIRED:

1585 ?? <Query>

1586 ?? <SubjectQuery>

1587 In addition, the following elements that are directly usable as part of SAML MAY be extended:

1588 ?? <Request>

1589 ?? <AuthenticationQuery>
1590 ?? <AuthorizationDecisionQuery>
1591 ?? <AttributeQuery>
1592 ?? <Response>

1593 **6.3. Use of Type Derivation and Substitution Groups**

1594 W3C XML Schema [**Schema1**] provides two principal mechanisms for specifying an element of an
1595 extended type: type derivation and substitution groups.

1596 For example, a <Statement> element can be assigned the type **NewStatementType** by means of
1597 the `xsi:type` attribute. For such an element to be schema-valid, **NewStatementType** needs to be
1598 derived from **StatementType**. The following example of a SAML assertion assumes that the
1599 extension schema (represented by the `new:` prefix) has defined this new type:

```
1600        <saml:Assertion ...>  
1601            <saml:Statement xsi:type="new:NewStatementType">  
1602            ...  
1603            </saml:Statement>  
1604        </saml:Assertion>
```

1605 Alternatively, the extension schema can define a <NewStatement> element that is a member of a
1606 substitution group that has <Statement> as a head element. For the substituted element to be
1607 schema-valid, it needs to have a type that matches or is derived from the head element's type. The
1608 following is an example of an extension schema fragment that defines this new element:

```
1609        <xsd:element "NewStatement" type="new:NewStatementType"  
1610            substitutionGroup="saml:Statement" />
```

1611 The substitution group declaration allows the <NewStatement> element to be used anywhere the
1612 SAML <Statement> element can be used. The following is an example of a SAML assertion that
1613 uses the extension element:

```
1614        <saml:Assertion ...>  
1615            <new:NewStatement>  
1616            ...  
1617            </new:NewStatement>  
1618        </saml:Assertion>
```

1619 The choice of extension method has no effect on the semantics of the XML document but does
1620 have implications for interoperability.

1621 The advantages of type derivation are as follows:

1622 ?? A document can be more fully interpreted by a parser that does not have access to the
1623 extension schema because a "native" SAML element is available.

1624 ?? At the time of writing, some W3C XML Schema validators do not support substitution
1625 groups, whereas the `xsi:type` attribute is widely supported.

1626 The advantage of substitution groups is that a document can be explained without the need to
1627 explain the functioning of the `xsi:type` attribute.

1628 **7. SAML-Defined Identifiers**

1629 The following sections define URI-based identifiers for common authentication protocols and
1630 actions.

1631 Where possible an existing URN is used to specify a protocol. In the case of IETF protocols the
1632 URN of the most current RFC that specifies the protocol is used. URI references created
1633 specifically for SAML have the initial stem:

1634 `urn:oasis:names:tc:SAML:1.0:`

1635 **7.1. Authentication Method Identifiers**

1636 The `<AuthenticationMethod>` and `<SubjectConfirmationMethod>` elements perform
1637 different functions within the SAML architecture, although both can refer to the same underlying
1638 mechanisms. `<AuthenticationMethod>` is a part of an Authentication Statement, which
1639 describes an authentication act which occurred in the past. The `<AuthenticationMethod>`
1640 indicates how that authentication was done. Note that the authentication statement does not
1641 provide the means to perform that authentication, such as a password, key or certificate.

1642 In contrast, `<SubjectConfirmationMethod>` is a part of the `<SubjectConfirmation>`, which
1643 is used to allow the Relying Party to confirm that the request or message came from the System
1644 Entity that corresponds to the Subject in the statement. The `<SubjectConfirmationMethod>`
1645 indicates the method that the Relying Party can use to do this in the future. This may or may not
1646 have any relationship to an authentication that was performed previously. Unlike the Authentication
1647 Method, the `<SubjectConfirmationMethod>` may be accompanied with some piece of
1648 information, such as a certificate or key, which will allow the Relying Party to perform the necessary
1649 check.

1650 Subject Confirmation Methods are defined in the SAML Profile or Profiles in which they are used
1651 **[SAMLBind]**. Additional methods may be added by defining new profiles or by private agreement.

1652 The following identifiers refer to SAML specified Authentication methods.

1653 **7.1.1. Password :**

1654 **URI:** `urn:oasis:names:tc:SAML:1.0:am:password`

1655 The authentication was performed by means of a password.

1656 **7.1.2. Kerberos**

1657 **URI:** `urn:ietf:rfc:1510`

1658 The authentication was performed by means of the Kerberos protocol **[RFC 1510]**, an instantiation
1659 of the Needham-Schroeder symmetric key authentication mechanism **[Needham78] [Needham78]**
1660 .

1661 **7.1.3. Secure Remote Password (SRP)**

1662 **URI:** `urn:ietf:rfc:2945`

1663 The authentication was performed by means of Secure Remote Password
1664 protocol as specified in **[RFC 2945]**

1665

7.1.4. Hardware Token

1666

URI: <urn:oasis:names:tc:SAML:1.0:am:HardwareToken>

1667

The authentication was performed by means of an unspecified hardware token.

1668

7.1.4.7.1.5. SSL/TLS Certificate Based Client Authentication:

1669

URI: <urn:ietf:rfc:2246>

1670

The authentication was performed using either the SSL or TLS protocol with certificate based client authentication. TLS is described in **[RFC 2246]**.

1671

1672

7.1.5.7.1.6. X.509 Public Key

1673

URI: <urn:oasis:names:tc:SAML:1.0:am:X509-PKI>

1674

The authentication was performed by some (unspecified) mechanism on a key authenticated by means of an X.509 PKI **[X.509][PKIX]**. It may have been one of the mechanisms for which a more specific identifier has been defined below.

1675

1676

1677

7.1.6.7.1.7. PGP Public Key

1678

URI: <urn:oasis:names:tc:SAML:1.0:am:PGP>

1679

The authentication was performed by some (unspecified) mechanism on a key authenticated by means of a PGP web of trust **[PGP]**. It may have been one of the mechanisms for which a more specific identifier has been defined below.

1680

1681

1682

7.1.7.7.1.8. SPKI Public Key

1683

URI: <urn:oasis:names:tc:SAML:1.0:am:SPKI>

1684

The authentication was performed by some (unspecified) mechanism on a key authenticated by means of a SPKI PKI **[SPKI]**. It may have been one of the mechanisms for which a more specific identifier has been defined below.

1685

1686

1687

7.1.8.7.1.9. XKMS Public Key

1688

URI: <urn:oasis:names:tc:SAML:1.0:am:XKMS>

1689

The authentication was performed by some (unspecified) mechanism on a key authenticated by means of a XKMS trust service **[XKMS]**. It may have been one of the mechanisms for which a more specific identifier has been defined below.

1690

1691

1692

7.1.9.7.1.10. XML Digital Signature

1693

URI: <urn:ietf:rfc:3075>

1694

The authentication was performed by means of an XML digital signature **[RFC 3075]**.

1695

7.2. Action Namespace Identifiers

1696

The following identifiers MAY be used in the `Namespace` attribute of the `<Action>` element (see Section 2.4.4.1) to refer to common sets of actions to perform on resources.

1697

1698 **7.2.1. Read/Write/Execute/Delete/Control:**

1699 **URI:** urn:oasis:names:tc:SAML:1.0:action:rwedc

1700 Defined actions:

1701 Read Write Execute Delete Control

1702 These actions are interpreted in the normal manner, i.e.

1703 Read

1704 The subject may read the resource

1705 Write

1706 The subject may modify the resource

1707 Execute

1708 The subject may execute the resource

1709 Delete

1710 The subject may delete the resource

1711 Control

1712 The subject may specify the access control policy for the resource

1713 **7.2.2. Read/Write/Execute/Delete/Control with Negation:**

1714 **URI:** urn:oasis:names:tc:SAML:1.0:action:rwedc-negation

1715 Defined actions:

1716 Read Write Execute Delete Control ~Read ~Write ~Execute ~Delete ~Control

1717 The actions specified in section 7.2.1 are interpreted in the same manner described there. Actions
1718 prefixed with a tilde ~ are negated permissions and are used to affirmatively specify that the stated
1719 permission is denied. Thus a subject described as being authorized to perform the action ~Read is
1720 affirmatively denied read permission.

1721 A SAML authority MUST NOT authorize both an action and its negated form.

1722 **7.2.3. Get/Head/Put/Post:**

1723 **URI:** urn:oasis:names:tc:SAML:1.0:action:ghpp

1724 Defined actions:

1725 GET HEAD PUT POST

1726 These actions bind to the corresponding HTTP operations. For example a subject authorized to
1727 perform the GET action on a resource is authorized to retrieve it.

1728 The GET and HEAD actions loosely correspond to the conventional read permission and the PUT
1729 and POST actions to the write permission. The correspondence is not exact however since a HTTP
1730 GET operation may cause data to be modified and a POST operation may cause modification to a
1731 resource other than the one specified in the request. For this reason a separate Action URI
1732 reference specifier is provided.

1733 **7.2.4. UNIX File Permissions:**

1734 **URI:** urn:oasis:names:tc:SAML:1.0:action:unix

1735 The defined actions are the set of UNIX file access permissions expressed in the numeric (octal)
1736 notation.

1737 The action string is a four digit numeric code:

1738 *extended user group world*

1739 Where the *extended* access permission has the value

1740 +2 if sgid is set

1741 +4 if suid is set

1742 The *user group* and *world* access permissions have the value

1743 +1 if execute permission is granted

1744 +2 if write permission is granted

1745 +4 if read permission is granted

1746 For example 0754 denotes the UNIX file access permission: user read, write and execute, group
1747 read and execute and world read.

8. SAML Schema Listings

1748

1749 The following sections contain complete listings of the assertion and protocol schemas for SAML.

8.1. Assertion Schema

1750

1751 Following is a complete listing of the SAML assertion schema [SAML-XSD].

```
1752 <?xml version="1.0" encoding="UTF-8"?>
1753 <!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-Baker
1754 (VeriSign Inc.) -->
1755 <schema
1756     targetNamespace="urn:oasis:names:tc:SAML:1.0:assertion"
1757     xmlns="http://www.w3.org/2001/XMLSchema"
1758     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
1759     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1760     elementFormDefault="unqualified">
1761     <import namespace="http://www.w3.org/2000/09/xmldsig#"
1762           schemaLocation="xmldsig-core-schema.xsd"/>
1763     <annotation>
1764         <documentation>draft-sstc-schema-assertion-3231.xsd</documentation>
1765     </annotation>
1766     <simpleType name="IDType">
1767         <restriction base="string"/>
1768     </simpleType>
1769     <simpleType name="IDReferenceType">
1770         <restriction base="string"/>
1771     </simpleType>
1772     <simpleType name="DecisionType">
1773         <restriction base="string">
1774             <enumeration value="Permit"/>
1775             <enumeration value="Deny"/>
1776             <enumeration value="Indeterminate"/>
1777         </restriction>
1778     </simpleType>
1779     <element name="AssertionIDReference" type="saml:IDReferenceType"/>
1780     <element name="Assertion" type="saml:AssertionType"/>
1781     <complexType name="AssertionType">
1782         <sequence>
1783             <element ref="saml:Conditions" minOccurs="0"/>
1784             <element ref="saml:Advice" minOccurs="0"/>
1785             <choice maxOccurs="unbounded">
1786                 <element ref="saml:Statement"/>
1787                 <element ref="saml:SubjectStatement"/>
1788                 <element ref="saml:AuthenticationStatement"/>
1789                 <element ref="saml:AuthorizationDecisionStatement"/>
1790                 <element ref="saml:AttributeStatement"/>
1791             </choice>
1792             <element ref="ds:Signature" minOccurs="0"/>
1793         </sequence>
1794         <attribute name="MajorVersion" type="integer" use="required"/>
1795         <attribute name="MinorVersion" type="integer" use="required"/>
1796         <attribute name="AssertionID" type="saml:IDType" use="required"/>
1797         <attribute name="Issuer" type="string" use="required"/>
1798         <attribute name="IssueInstant" type="dateTime" use="required"/>
1799     </complexType>
1800     <element name="Conditions" type="saml:ConditionsType"/>
1801     <complexType name="ConditionsType">
1802         <choice minOccurs="0" maxOccurs="unbounded">
1803             <element ref="saml:AudienceRestrictionCondition"/>
1804             <element ref="saml:Condition"/>
```

```

1805     </choice>
1806     <attribute name="NotBefore" type="dateTime" use="optional"/>
1807     <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>
1808 </complexType>
1809 <element name="Condition" type="saml:ConditionAbstractType"/>
1810 <complexType name="ConditionAbstractType" abstract="true"/>
1811 <element name="AudienceRestrictionCondition"
1812     type="saml:AudienceRestrictionConditionType"/>
1813 <complexType name="AudienceRestrictionConditionType">
1814     <complexContent>
1815         <extension base="saml:ConditionAbstractType">
1816             <sequence>
1817                 <element ref="saml:Audience" maxOccurs="unbounded"/>
1818             </sequence>
1819         </extension>
1820     </complexContent>
1821 </complexType>
1822 <element name="Audience" type="anyURI"/>
1823 <element name="Advice" type="saml:AdviceType"/>
1824 <complexType name="AdviceType">
1825     <choice minOccurs="0" maxOccurs="unbounded">
1826         <element ref="saml:AssertionIDReference"/>
1827         <element ref="saml:Assertion"/>
1828         <any namespace="##other" processContents="lax"/>
1829     </choice>
1830 </complexType>
1831 <element name="Statement" type="saml:StatementAbstractType"/>
1832 <complexType name="StatementAbstractType" abstract="true"/>
1833 <element name="SubjectStatement" type="saml:SubjectStatementAbstractType"/>
1834 <complexType name="SubjectStatementAbstractType" abstract="true">
1835     <complexContent>
1836         <extension base="saml:StatementAbstractType">
1837             <sequence>
1838                 <element ref="saml:Subject"/>
1839             </sequence>
1840         </extension>
1841     </complexContent>
1842 </complexType>
1843 <element name="Subject" type="saml:SubjectType"/>
1844 <complexType name="SubjectType">
1845     <choice>
1846         <sequence>
1847             <element ref="saml:NameIdentifier"/>
1848             <element ref="saml:SubjectConfirmation" minOccurs="0"/>
1849         </sequence>
1850         <element ref="saml:SubjectConfirmation"/>
1851     </choice>
1852 </complexType>
1853 <element name="NameIdentifier" type="saml:NameIdentifierType"/>
1854 <complexType name="NameIdentifierType">
1855     <simpleContent>
1856         <extension base="string">
1857             <attribute name="NameQualifier" type="string" use="optional"/>
1858             <attribute name="Format" type="anyURI" use="optional"/>
1859         </extension>
1860     </simpleContent>
1861 </complexType>
1862 <element name="SubjectConfirmation" type="saml:SubjectConfirmationType"/>
1863 <complexType name="SubjectConfirmationType">
1864     <sequence>
1865         <element ref="saml:ConfirmationMethod" maxOccurs="unbounded"/>
1866         <element ref="saml:SubjectConfirmationData" minOccurs="0"/>
1867         <element ref="ds:KeyInfo" minOccurs="0"/>

```

```

1868     </sequence>
1869 </complexType>
1870 <element name="SubjectConfirmationData" type="anyType" />
1871 <element name="ConfirmationMethod" type="anyURI" />
1872 <element name="AuthenticationStatement"
1873     type="saml:AuthenticationStatementType" />
1874 <complexType name="AuthenticationStatementType">
1875     <complexContent>
1876         <extension base="saml:SubjectStatementAbstractType">
1877             <sequence>
1878                 <element ref="saml:SubjectLocality" minOccurs="0" />
1879                 <element ref="saml:AuthorityBinding"
1880                     minOccurs="0" maxOccurs="unbounded" />
1881             </sequence>
1882             <attribute name="AuthenticationMethod" type="anyURI" />
1883             <attribute name="AuthenticationInstant" type="dateTime" />
1884         </extension>
1885     </complexContent>
1886 </complexType>
1887 <element name="SubjectLocality"
1888     type="saml:SubjectLocalityType" />
1889 <complexType name="SubjectLocalityType">
1890     <attribute name="IPAddress" type="string" use="optional" />
1891     <attribute name="DNSAddress" type="string" use="optional" />
1892 </complexType>
1893 <element name="AuthorityBinding" type="saml:AuthorityBindingType" />
1894 <complexType name="AuthorityBindingType">
1895     <attribute name="AuthorityKind" type="QName" use="required" />
1896     <attribute name="Location" type="anyURI" use="required" />
1897     <attribute name="Binding" type="anyURI" use="required" />
1898 </complexType>
1899 <element name="AuthorizationDecisionStatement"
1900 type="saml:AuthorizationDecisionStatementType" />
1901 <complexType name="AuthorizationDecisionStatementType">
1902     <complexContent>
1903         <extension base="saml:SubjectStatementAbstractType">
1904             <sequence>
1905                 <element ref="saml:Action" maxOccurs="unbounded" />
1906                 <element ref="saml:Evidence" minOccurs="0" />
1907             </sequence>
1908             <attribute name="Resource" type="anyURI" use="required" />
1909             <attribute name="Decision" type="saml:DecisionType" use="required" />
1910         </extension>
1911     </complexContent>
1912 </complexType>
1913 <element name="Action" type="saml:ActionType" />
1914 <complexType name="ActionType">
1915     <simpleContent>
1916         <extension base="string">
1917             <attribute name="Namespace" type="anyURI" />
1918         </extension>
1919     </simpleContent>
1920 </complexType>
1921 <element name="Evidence" type="saml:EvidenceType" />
1922 <complexType name="EvidenceType">
1923     <choice maxOccurs="unbounded">
1924         <element ref="saml:AssertionIDReference" />
1925         <element ref="saml:Assertion" />
1926     </choice>
1927 </complexType>
1928 <element name="AttributeStatement" type="saml:AttributeStatementType" />
1929 <complexType name="AttributeStatementType">
1930     <complexContent>

```

```

1931     <extension base="saml:SubjectStatementAbstractType">
1932         <sequence>
1933             <element ref="saml:Attribute" maxOccurs="unbounded"/>
1934         </sequence>
1935     </extension>
1936 </complexContent>
1937 </complexType>
1938 <element name="AttributeDesignator" type="saml:AttributeDesignatorType"/>
1939 <complexType name="AttributeDesignatorType">
1940     <attribute name="AttributeName" type="string" use="required"/>
1941     <attribute name="AttributeNamespace" type="anyURI" use="required"/>
1942 </complexType>
1943 <element name="Attribute" type="saml:AttributeType"/>
1944 <complexType name="AttributeType">
1945     <complexContent>
1946         <extension base="saml:AttributeDesignatorType">
1947             <sequence>
1948                 <element ref="saml:AttributeValue" maxOccurs="unbounded"/>
1949             </sequence>
1950         </extension>
1951     </complexContent>
1952 </complexType>
1953 <element name="AttributeValue" type="saml:anyType"/>
1954 </schema>

```

1955 8.2. Protocol Schema

1956 Following is a complete listing of the SAML protocol schema [SAML-XSD].

```

1957 <?xml version="1.0" encoding="UTF-8"?>
1958 <!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-Baker
1959 (VeriSign Inc.) -->
1960 <schema
1961     targetNamespace="urn:oasis:names:tc:SAML:1.0:protocol"
1962     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1963     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
1964     xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
1965     xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
1966     <import
1967         namespace="urn:oasis:names:tc:SAML:1.0:assertion"
1968         schemaLocation="draft-sstc-schema-assertion-3231.xsd"/>
1969     <import namespace="http://www.w3.org/2000/09/xmldsig#"
1970         schemaLocation="xmldsig-core-schema.xsd"/>
1971     <annotation>
1972         <documentation>draft-sstc-schema-protocol-3231.xsd</documentation>
1973     </annotation>
1974     <complexType name="RequestAbstractType" abstract="true">
1975         <sequence>
1976             <element ref="samlp:RespondWith"
1977                 minOccurs="0" maxOccurs="unbounded"/>
1978             <element ref="ds:Signature" minOccurs="0"/>
1979         </sequence>
1980         <attribute name="RequestID" type="saml:IDType" use="required"/>
1981         <attribute name="MajorVersion" type="integer" use="required"/>
1982         <attribute name="MinorVersion" type="integer" use="required"/>
1983         <attribute name="IssueInstant" type="dateTime" use="required"/>
1984     </complexType>
1985     <element name="RespondWith" type="QName"/>
1986     <element name="Request" type="samlp:RequestType"/>
1987     <complexType name="RequestType">
1988         <complexContent>
1989             <extension base="samlp:RequestAbstractType">
1990                 <choice>

```

```

1991         <element ref="sampl:Query" />
1992         <element ref="sampl:SubjectQuery" />
1993         <element ref="sampl:AuthenticationQuery" />
1994         <element ref="sampl:AttributeQuery" />
1995         <element ref="sampl:AuthorizationDecisionQuery" />
1996         <element ref="saml:AssertionID" maxOccurs="unbounded" />
1997         <element ref="sampl:AssertionArtifact" maxOccurs="unbounded" />
1998     </choice>
1999 </extension>
2000 </complexContent>
2001 </complexType>
2002 <element name="AssertionArtifact" type="string" />
2003 <element name="Query" type="sampl:QueryAbstractType" />
2004 <complexType name="QueryAbstractType" abstract="true" />
2005 <element name="SubjectQuery" type="sampl:SubjectQueryAbstractType" />
2006 <complexType name="SubjectQueryAbstractType" abstract="true">
2007     <complexContent>
2008         <extension base="sampl:QueryAbstractType">
2009             <sequence>
2010                 <element ref="saml:Subject" />
2011             </sequence>
2012         </extension>
2013     </complexContent>
2014 </complexType>
2015 <element name="AuthenticationQuery" type="sampl:AuthenticationQueryType" />
2016 <complexType name="AuthenticationQueryType">
2017     <complexContent>
2018         <extension base="sampl:SubjectQueryAbstractType">
2019             <attribute name="AuthenticationMethod" type="anyURI" />
2020         </extension>
2021     </complexContent>
2022 </complexType>
2023 <element name="AttributeQuery" type="sampl:AttributeQueryType" />
2024 <complexType name="AttributeQueryType">
2025     <complexContent>
2026         <extension base="sampl:SubjectQueryAbstractType">
2027             <sequence>
2028                 <element ref="saml:AttributeDesignator"
2029                     minOccurs="0" maxOccurs="unbounded" />
2030             </sequence>
2031             <attribute name="Resource" type="anyURI" use="optional" />
2032         </extension>
2033     </complexContent>
2034 </complexType>
2035 <element name="AuthorizationDecisionQuery"
2036     type="sampl:AuthorizationDecisionQueryType" />
2037 <complexType name="AuthorizationDecisionQueryType">
2038     <complexContent>
2039         <extension base="sampl:SubjectQueryAbstractType">
2040             <sequence>
2041                 <element ref="saml:Action" maxOccurs="unbounded" />
2042                 <element ref="saml:Evidence"
2043                     minOccurs="0" maxOccurs="unbounded" />
2044             </sequence>
2045             <attribute name="Resource" type="anyURI" use="required" />
2046         </extension>
2047     </complexContent>
2048 </complexType>
2049 <complexType name="ResponseAbstractType" abstract="true">
2050     <sequence>
2051         <element ref="ds:Signature" minOccurs="0" />
2052     </sequence>
2053     <attribute name="ResponseID" type="saml:IDType" use="required" />

```

```

2054     <attribute name="InResponseTo" type="saml:IDReferenceType"
2055         use="optional"/>
2056     <attribute name="MajorVersion" type="integer" use="required"/>
2057     <attribute name="MinorVersion" type="integer" use="required"/>
2058     <attribute name="IssueInstant" type="dateTime" use="required"/>
2059     <attribute name="Recipient" type="anyURI" use="optional"/>
2060 </complexType>
2061 <element name="Response" type="sampl:ResponseType"/>
2062 <complexType name="ResponseType">
2063     <complexContent>
2064         <extension base="sampl:ResponseAbstractType">
2065             <sequence>
2066                 <element ref="sampl:Status"/>
2067                 <element ref="saml:Assertion"
2068                     minOccurs="0" maxOccurs="unbounded"/>
2069             </sequence>
2070         </extension>
2071     </complexContent>
2072 </complexType>
2073 <element name="Status" type="sampl:StatusType"/>
2074 <complexType name="StatusType">
2075     <sequence>
2076         <element ref="sampl:StatusCode"/>
2077         <element ref="sampl:StatusMessage"
2078             minOccurs="0" maxOccurs="unbounded"/>
2079         <element ref="sampl:StatusDetail" minOccurs="0"/>
2080     </sequence>
2081 </complexType>
2082 <element name="StatusCode" type="sampl:StatusCodeType"/>
2083 <complexType name="StatusCodeType">
2084     <sequence>
2085         <element ref="sampl:StatusCode" minOccurs="0"/>
2086     </sequence>
2087     <attribute name="Value" type="QName" use="required"/>
2088 </complexType>
2089 <element name="StatusMessage" type="string"/>
2090 <element name="StatusDetail" type="sampl:StatusDetailType"/>
2091 <complexType name="StatusDetailType">
2092     <sequence>
2093         <any namespace="##any"
2094             processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2095     </sequence>
2096 </complexType>
2097 </schema>
2098

```

9. References

2099

- 2100 [Kern-84] B. Kernighan, Rob Pike *The UNIX Programming Environment*, (March
2101 1984) Prentice Hall Computer Books;
- 2102 [Needham78] R. Needham et al., *Using Encryption for Authentication in Large Networks*
2103 *of Computers*, Communications of the ACM, Vol. 21 (12), pp. 993-999,
2104 December 1978.
- 2105 [PGP] Atkins, D., Stallings, W. and P. Zimmermann, *PGP Message Exchange*
2106 *Formats*, RFC 1991, August 1996.
- 2107 [PKCS1] B. Kaliski, *PKCS #1: RSA Encryption Version 2.0*, RSA Laboratories, also
2108 IETF RFC 2437, October 1998. <http://www.ietf.org/rfc/rfc2437.txt>
- 2109 [PKCS7] B. Kaliski., *PKCS #7: Cryptographic Message Syntax, Version 1.5* RFC
2110 2315, March 1998.
- 2111 [PKIX] R. Housley, W. Ford, W. Polk, D. Solo. *Internet X.509 Public Key*
2112 *Infrastructure Certificate and CRL Profile*. RFC 2459, January 1999.
- 2113 [RFC 1510] J. Kohl, C. Neuman. *The Kerberos Network Authentication Requestor (V5)*.
2114 September 1993. <http://www.ietf.org/rfc/rfc1510.txt>
- 2115 [RFC 2104] H. Krawczyk et al., *HMAC: Keyed Hashing for Message Authentication*,
2116 <http://www.ietf.org/rfc/rfc2104.txt>, IETF RFC 2104, February 1997.
- 2117 [RFC 2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
2118 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997
- 2119 [RFC 2246] T. Dierks, C. Allen. *The TLS Protocol Version 1.0*. January 1999.
2120 <http://www.ietf.org/rfc/rfc2246.txt>
- 2121 [RFC 2396] T. Berners-Lee et. al., *Uniform Resource Identifiers (URI): Generic Syntax*
2122 <http://www.ietf.org/rfc/rfc2396.txt> IETF?
- 2123 [RFC 2630] R. Housley. *Cryptographic Message Syntax*. June 1999.
2124 <http://www.ietf.org/rfc/rfc630.txt>
- 2125 [RFC 2648] R. Moats. *A URN Namespace for IETF Documents*. August 1999.
2126 <http://www.ietf.org/rfc/rfc2648.txt>
- 2127 [RFC 2945] T. Wu, *The SRP Authentication and Key Exchange System*, September
2128 2000, <http://www.ietf.org/rfc/rfc2945.txt> [RFC 3075] D. Eastlake, J.
2129 Reagle, D. Solo. *XML -Signature Syntax and Processing*. March 2001.
2130 <http://www.ietf.org/rfc/rfc3075.txt>
- 2131 [SAMLBind] P. Mishra et al., *Bindings and Profiles for the OASIS Security Assertion*
2132 *Markup Language (SAML)*, [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-bindings-model-07.pdf)
2133 [open.org/committees/security/docs/draft-sstc-bindings-model-07.pdf](http://www.oasis-open.org/committees/security/docs/draft-sstc-bindings-model-07.pdf),
2134 OASIS, December 2001.
- 2135 [SAMLConform] **TBS**
- 2136 [SAMLGloss] J. Hodges et al., *Glossary for the OASIS Security Assertion Markup*
2137 *Language (SAML)*, [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-glossary-02.pdf)
2138 [open.org/committees/security/docs/draft-sstc-glossary-02.pdf](http://www.oasis-open.org/committees/security/docs/draft-sstc-glossary-02.pdf), OASIS,
2139 December 2001.
- 2140 [SAMLXSD] P. Hallam-Baker et al., *SAML protocol schema*, [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-21.xsd)
2141 [open.org/committees/security/docs/draft-sstc-schema-protocol-21.xsd](http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-21.xsd),
2142 OASIS, December 2001.
- 2143 [SAMLSecure] **TBS**

2144	[SAML-XSD]	P. Hallam-Baker et al., <i>SAML assertion schema</i> , http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-21.xsd , OASIS, December 2001.
2145		
2146		
2147	[Schema1]	H. S. Thompson et al., <i>XML Schema Part 1: Structures</i> , http://www.w3.org/TR/xmlschema-1/ , World Wide Web Consortium Recommendation, May 2001.
2148		
2149		
2150	[Schema2]	P. V. Biron et al., <i>XML Schema Part 2: Datatypes</i> , http://www.w3.org/TR/xmlschema-2 , World Wide Web Consortium Recommendation, May 2001.
2151		
2152		
2153	[SPKI]	C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen. <i>SPKI Certificate Theory</i> , RFC 2693, September 1999.
2154		
2155	[UNICODE-C]	M. Davis, M. J. Dürst, http://www.unicode.org/unicode/reports/tr15/tr15-21.html , UNICODE Consortium
2156		
2157	[W3C-CHAR]	M. J. Dürst, <i>Requirements for String Identity Matching and String Indexing</i> http://www.w3.org/TR/WD-charreq , World Wide Web Consortium.
2158		
2159	[W3C-CharMod]	M. J. Dürst, <i>Unicode Normalization Forms</i> http://www.w3.org/TR/charmod/ , World Wide Web Consortium.
2160		
2161	[X.500]	ITU-T Recommendation X.501: <i>Information Technology - Open Systems Interconnection - The Directory: Models</i> , 1993.
2162		
2163	[XKMS]	W. Ford, P. Hallam-Baker, B. Fox, B. Dillaway, B. LaMacchia, J. Epstein, J. Lapp, XML Key Management Specification (XKMS), W3C Note 30 March 2001, http://www.w3.org/TR/xkms/
2164		
2165		
2166	[XML]	T. Bray et. al. <i>Extensible Markup Language (XML) 1.0 (Second Edition)</i> , http://www.w3.org/TR/REC-xml , World Wide Web Consortium.
2167		
2168	[XMLEnc]	<i>XML Encryption Specification</i> , In development.
2169	[XMLSig]	D. Eastlake et al., <i>XML-Signature Syntax and Processing</i> , http://www.w3.org/TR/xmlsig-core/ , World Wide Web Consortium.
2170		
2171	[XMLSig-XSD]	XML Signature Schema available from http://www.w3.org/TR/2000/CR-xmlsig-core-20001031/xmlsig-core-schema.xsd .
2172		
2173	[XTAML]	P. Hallam-Baker, <i>XML Trust Axiom Markup Language 1.0</i> , http://www.xmltrustcenter.org/ , VeriSign Inc. September 2001.
2174		

2175

10. Acknowledgements

2176
2177

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

2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203

- Allen Rogers, Authentica
- Irving Reid, Baltimore Technologies
- Krishna Sankar, Cisco Systems Inc
- Simon Godik, Crosslogix
- Gil Pilz, E2open
- Hal Lockhart, Entegrity Solutions
- Carlisle Adams, Entrust Inc.
- Robert Griffin, Entrust Inc.
- Don Flinn, Hitachi
- Joe Pato, Hewlett-Packard (co-Chair)
- Jason Rouault, Hewlett-Packard
- Marc Chanliau, Netegrity
- Chris McLaren, Netegrity
- Prateek Mishra, Netegrity
- Charles Knouse, Oblix
- Steve Anderson, OpenNetwork
- Rob Philpott, RSA Security
- Jahan Moreh, Sigaba
- Bhavna Bhatnagar, Sun Microsystems
- Jeff Hodges, Sun Microsystems (co-Chair)
- Eve Maler, Sun Microsystems (Former Chair)
- Aravindan Ranganathan, Sun Microsystems
- Emily Xu, Sun Microsystems
- Bob Morgan, University of Washington
- Phillip Hallam-Baker, VeriSign Inc.

2204

The editors would also like to thank the following people for their contributions:

2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226

- Stephen Farrell, Baltimore Technologies
- David Orchard, BEA Systems
- Tim Moses, Entrust Inc.
- Nigel Edwards, Hewlett-Packard
- Marc Chanliau, Netegrity
- Scott Cantor, The Ohio State University
- Darren Platt, Formerly with RSA Security
- Bob Blakley IBM Tivoli Software
- Marlena Erdos, Tivoli

2227
2228

Appendix A. Notices

2229

2230 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
2231 that might be claimed to pertain to the implementation or use of the technology described in this
2232 document or the extent to which any license under such rights might or might not be available;
2233 neither does it represent that it has made any effort to identify any such rights. Information on
2234 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
2235 website. Copies of claims of rights made available for publication and any assurances of licenses to
2236 be made available, or the result of an attempt made to obtain a general license or permission for
2237 the use of such proprietary rights by implementors or users of this specification, can be obtained
2238 from the OASIS Executive Director.

2239 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
2240 applications, or other proprietary rights which may cover technology that may be required to
2241 implement this specification. Please address the information to the OASIS Executive Director.

2242 Copyright © The Organization for the Advancement of Structured Information Standards [OASIS]
2243 2001. All Rights Reserved.

2244 This document and translations of it may be copied and furnished to others, and derivative works
2245 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
2246 published and distributed, in whole or in part, without restriction of any kind, provided that the above
2247 copyright notice and this paragraph are included on all such copies and derivative works. However,
2248 this document itself may not be modified in any way, such as by removing the copyright notice or
2249 references to OASIS, except as needed for the purpose of developing OASIS specifications, in
2250 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights
2251 document must be followed, or as required to translate it into languages other than English.

2252 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
2253 successors or assigns.

2254 This document and the information contained herein is provided on an "AS IS" basis and OASIS
2255 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
2256 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY
2257 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
2258 PARTICULAR PURPOSE.