



1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35

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

**Document identifier:** draft-sstc-core-26

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

**Publication date:** January 10th 2002

**Maturity Level:** Committee Working Draft

**Send comments to:** [security-requestors-comment@lists.oasis-open.org](mailto: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-services-comment-request@lists.oasis-open.org](mailto:security-services-comment-request@lists.oasis-open.org) or [security-requestors-comment-request@lists.oasis-open.org](mailto: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

**Contributors:**

Carlisle Adams, Entrust  
Scott Cantor, The Ohio State University  
Marc Chanliau, Netegrity  
Nigel Edwards, Hewlett-Packard  
Marlena Erdos, Tivoli  
Stephen Farrell, Baltimore Technologies  
Simon Godik, Crosslogic  
Jeff Hodges, Oblix  
Charles Knouse, Oblix  
Chris McLaren, Netegrity  
Prateek Mishra, Netegrity  
RL "Bob" Morgan, University of Washington  
Tim Moses, Entrust  
David Orchard, BEA  
Joe Pato, Hewlett Packard  
Darren Platt, RSA [Security](#)  
Irving Reid, Baltimore  
Krishna Sankar, Cisco Systems Inc

35		
36	ASSERTIONS AND PROTOCOL FOR THE OASIS SECURITY ASSERTION MARKUP	
37	LANGUAGE (SAML)	1
38	<b>1. INTRODUCTION</b>	<b>6</b>
39	1.1. NOTATION	6
40	1.2. SCHEMA ORGANIZATION AND NAMESPACES	6
41	1.3. SAML CONCEPTS (NON-NORMATIVE)	7
42	1.3.1. <i>Overview</i>	7
43	1.3.2. <i>SAML and URI-Based Identifiers</i>	<del>9</del>
44	1.3.3. <i>SAML and Extensibility</i>	9
45	<b>2. SAML ASSERTIONS</b>	<b>10</b>
46	2.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS	10
47	2.2. SIMPLE TYPES	10
48	2.2.1. <i>Simple Types IDType and IDReferenceType</i>	10
49	2.2.2. <i>Simple Type DecisionType</i>	11
50	2.3. ASSERTIONS	<del>12+1</del>
51	2.3.1. <i>Element &lt;AssertionSpecifier&gt;</i>	<del>12+1</del>
52	2.3.2. <i>Element &lt;AssertionID&gt;</i>	12
53	2.3.3. <i>Element &lt;Assertion&gt;</i>	12
54	2.3.3.1. <i>Element &lt;Conditions&gt;</i>	13
55	2.3.3.1.1. <i>Attributes NotBefore and NotOnOrAfter</i>	14
56	2.3.3.1.2. <i>Element &lt;Condition&gt;</i>	<del>15+4</del>
57	2.3.3.1.3. <i>Elements &lt;AudienceRestrictionCondition&gt; and &lt;Audience&gt;</i>	<del>15+4</del>
58	2.3.3.1.4. <i>Elements &lt;TargetRestrictionCondition&gt; and &lt;Target&gt;</i>	<del>16+5</del>
59	2.3.3.2. <i>Elements &lt;Advice&gt; and &lt;AdviceElement&gt;</i>	<del>16+5</del>
60	2.4. STATEMENTS	<del>17+6</del>
61	2.4.1. <i>Element &lt;Statement&gt;</i>	<del>17+6</del>
62	2.4.2. <i>Element &lt;SubjectStatement&gt;</i>	<del>17+6</del>

63	2.4.2.1. Element <Subject>	17
64	2.4.2.2. Element <NameIdentifier>	<del>1847</del>
65	2.4.2.3. Elements <SubjectConfirmation>, <ConfirmationMethod>, and <SubjectConfirmationData>	<del>1847</del>
66	<b>2.4.3. Element &lt;AuthenticationStatement&gt;</b>	<del>1948</del>
67	2.4.3.1. Element <AuthenticationLocality>	19
68	2.4.3.2. Element <AuthorityBinding>	<del>2049</del>
69	<b>2.4.4. Element &lt;AuthorizationDecisionStatement&gt;</b>	20
70	2.4.4.1. Elements <Actions> and <Action>	<del>2120</del>
71	2.4.4.2. Element <Evidence>	<del>2221</del>
72	<b>2.4.5. Element &lt;AttributeStatement&gt;</b>	<del>2221</del>
73	2.4.5.1. Elements <AttributeDesignator> and <Attribute>	<del>2221</del>
74	2.4.5.1.1 Element <AttributeValue>	<del>2322</del>
75	<b>3. SAML PROTOCOL</b>	<b>23</b>
76	<b>3.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS</b>	<del>2423</del>
77	<b>3.2. REQUESTS</b>	<del>2423</del>
78	<b>3.2.1. Complex Type RequestAbstractType</b>	<del>2423</del>
79	3.2.1.1. Element <RespondWith>	<del>2524</del>
80	<b>3.2.2. Element &lt;Request&gt;</b>	<del>2625</del>
81	<b>3.2.3. Element &lt;AssertionArtifact&gt;</b>	<del>2725</del>
82	<b>3.3. QUERIES</b>	<del>2726</del>
83	<b>3.3.1. Element &lt;Query&gt;</b>	<del>2726</del>
84	<b>3.3.2. Element &lt;SubjectQuery&gt;</b>	<del>2726</del>
85	<b>3.3.3. Element &lt;AuthenticationQuery&gt;</b>	<del>2726</del>
86	<b>3.3.4. Element &lt;AttributeQuery&gt;</b>	<del>2827</del>
87	<b>3.3.5. Element &lt;AuthorizationDecisionQuery&gt;</b>	<del>2827</del>
88	<b>3.4. RESPONSES</b>	<del>2928</del>
89	<b>3.4.1. Complex Type ResponseAbstractType</b>	<del>2928</del>
90	<b>3.4.2. Element &lt;Response&gt;</b>	<del>3028</del>
91	<b>3.4.3. Element &lt;Status&gt;</b>	<del>3029</del>
92	3.4.3.1. Element <StatusCode>	<del>3129</del>

93	3.4.3.2. Element <SubStatusCode>	<del>3130</del>
94	3.4.3.3. Element <StatusMessage>	<del>3231</del>
95	3.4.3.4. Element <StatusDetail>	<del>3231</del>
96	<b>4. SAML VERSIONING</b>	<del>3231</del>
97	4.1. ASSERTION VERSION	<del>3432</del>
98	4.2. REQUEST VERSION	<del>3432</del>
99	4.3. RESPONSE VERSION	<del>3533</del>
100	<b>5. SAML &amp; XML-SIGNATURE SYNTAX AND PROCESSING</b>	<del>3634</del>
101	5.1. SIGNING ASSERTIONS	<del>3634</del>
102	5.2. REQUEST /RESPONSE SIGNING	<del>3735</del>
103	5.3. SIGNATURE INHERITANCE	<del>3735</del>
104	5.3.1. <i>Rationale</i>	<del>3735</del>
105	5.3.2. <i>Rules for SAML Signature Inheritance</i>	<del>3735</del>
106	5.4. XML SIGNATURE PROFILE	<del>3735</del>
107	5.4.1. <i>Signing formats</i>	<del>3735</del>
108	5.4.2. <i>CanonicalizationMethod</i>	<del>3835</del>
109	5.4.3. <i>Transforms</i>	<del>3836</del>
110	5.4.4. <i>KeyInfo</i>	<del>3836</del>
111	5.4.5. <i>Binding between statements in a multi-statement assertion</i>	<del>3836</del>
112	<b>6. SAML EXTENSIONS</b>	<del>3837</del>
113	6.1. ASSERTION SCHEMA EXTENSION	<del>3937</del>
114	6.2. PROTOCOL SCHEMA EXTENSION	<del>3937</del>
115	6.3. USE OF TYPE DERIVATION AND SUBSTITUTION GROUPS	<del>4038</del>
116	<b>7. SAML-DEFINED IDENTIFIERS</b>	<del>4139</del>

117	<b>7.1. CONFIRMATION METHOD IDENTIFIERS</b>	<del>4139</del>
118	<b>7.1.1. SAML Artifact:</b>	<del>4139</del>
119	<b>7.1.2. SAML Artifact (SHA-1):</b>	<del>4139</del>
120	<b>7.1.3. Holder of Key:</b>	<del>4139</del>
121	<b>7.1.4. Sender Vouches:</b>	<del>4139</del>
122	<b>7.1.5. Password (Pass-Through):</b>	<del>4239</del>
123	<b>7.1.6. Password (One-Way-Function SHA-1):</b>	<del>4240</del>
124	<b>7.1.7. Kerberos</b>	<del>4240</del>
125	<b>7.1.8. SSL/TLS Certificate Based Client Authentication:</b>	<del>4240</del>
126	<b>7.1.9. Object Authenticator (SHA-1):</b>	<del>4240</del>
127	<b>7.1.10. PKCS#7</b>	<del>4240</del>
128	<b>7.1.11. Cryptographic Message Syntax</b>	<del>4344</del>
129	<b>7.1.12. XML Digital Signature</b>	<del>4344</del>
130	<b>7.2. ACTION NAMESPACE IDENTIFIERS</b>	<del>4344</del>
131	<b>7.2.1. Read/Write/Execute/Delete/Control:</b>	<del>4344</del>
132	<b>7.2.2. Read/Write/Execute/Delete/Control with Negation:</b>	<del>4444</del>
133	<b>7.2.3. Get/Head/Put/Post:</b>	<del>4442</del>
134	<b>7.2.4. UNIX File Permissions:</b>	<del>4442</del>
135	<b>8. SAML SCHEMA LISTINGS</b>	<del>4543</del>
136	<b>8.1. ASSERTION SCHEMA</b>	<del>4543</del>
137	<b>8.2. PROTOCOL SCHEMA</b>	<del>4946</del>
138	<b>9. REFERENCES</b>	<del>5250</del>
139	<b>APPENDIX A. NOTICES</b>	<del>5452</del>
140		

# 1. Introduction

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 [SAML-PSD] are available.

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

## 1.1. Notation

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.

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 [RFC2119]:

*"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)"*

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.

Listings of SAML schemas appear like this.

Example code listings appear like this.

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:

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

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

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

?? 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.

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

## 1.2. Schema Organization and Namespaces

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

<http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-26.xsd>

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

<http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-26.xsd>

draft-sstc-core-2626 ————— 6 ————— 6

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

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

186 <http://www.w3.org/2000/09/xmlsig#>

187 **1.2.1. The XML Signature element `<ds:KeyInfo>`, defined in [XMLSig] §4.4, is**  
188 **of particular interest in SAML Time Values.**

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

191 SAML applications SHOULD NOT rely on other applications supporting time resolution finer than  
192 milliseconds. Implementations MUST NOT generate time instants that specify leap seconds.

193 **1.2.2. Comparing SAML values**

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

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

206 Applications that compare data received in SAML documents to data from external sources MUST  
207 take into account the normalization rules specified for XML. Text contained within elements is  
208 normalized so that line endings are represented using linefeed characters (ASCII code 10<sub>Decimal</sub>), as  
209 described in section 2.11 of the XML Recommendation [**XML**]. Attribute values defined as strings  
210 (or types derived from strings) are normalized as described in section 3.3.3 [**XML**] all white space  
211 characters are replaced with blanks (ASCII code 32<sub>Decimal</sub>).

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

215 **1.3. SAML Concepts (Non-Normative)**

216 This section is informative only and is superseded by any contradicting information in the normative  
217 text in Sections 1.2 and following. A glossary of SAML terms and concepts [**SAMLGloss**] is  
218 available.

219 **1.3.1. Overview**

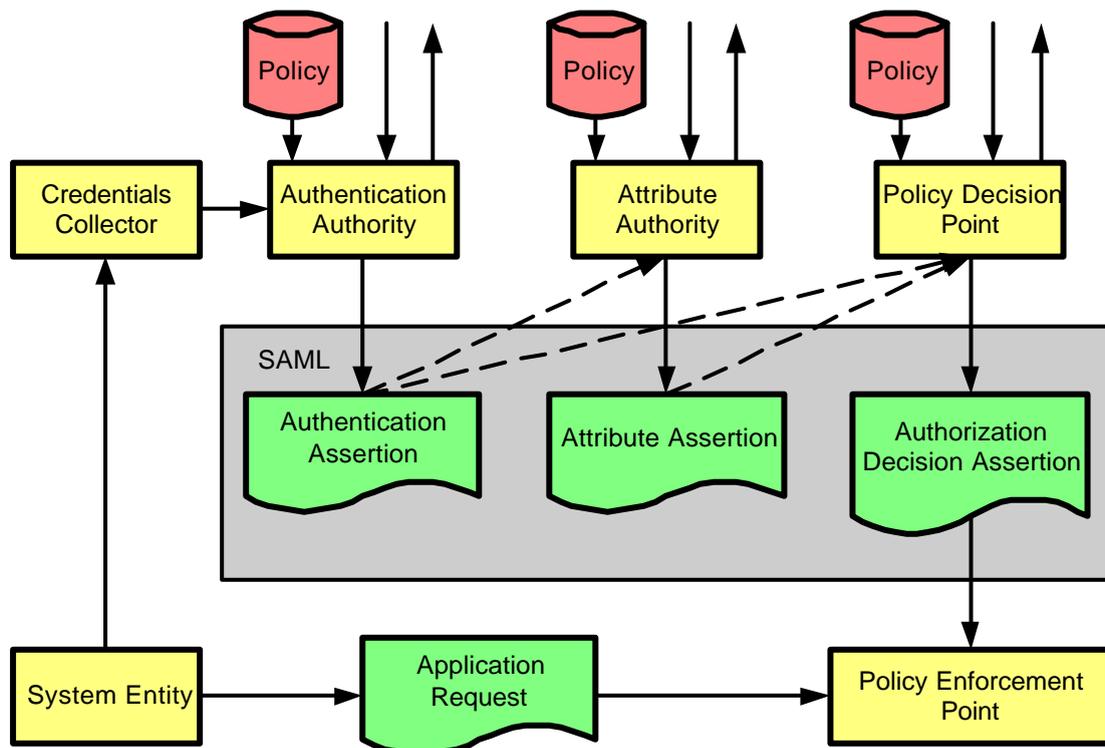
220 The Security Assertion Markup Language (SAML) is an XML-based framework for exchanging  
221 security information. This security information is expressed in the form of assertions about subjects,  
222 where a subject is an entity (either human or computer) that has an identity in some security 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

225 Assertions can convey information about authentication acts performed by subjects, attributes of  
 226 subjects, and authorization decisions about whether subjects are allowed to access certain  
 227 resources. Assertions are represented as XML constructs and have a nested structure, whereby a  
 228 single assertion might contain several different internal statements about authentication,  
 229 authorization, and attributes. Note that authentication assertions merely describe acts of  
 230 authentication that happened previously; ~~checking and revoking of credentials is outside the scope~~  
 231 ~~of this version of SAML.~~

232 Assertions are issued by SAML authorities, namely, authentication authorities, attribute authorities,  
 233 and policy decision points. SAML ~~provides defines~~ a protocol by which clients can request  
 234 assertions from SAML authorities and get a response from them. This protocol, consisting of XML-  
 235 based request and response message formats, can be bound to many different underlying  
 236 communications and transport protocols; SAML currently defines one binding, to SOAP over HTTP.  
 237 ~~It is possible to produce and consume SAML assertions without using the SAML protocol, although~~  
 238 ~~interoperability is likely to be harmed in this case.~~

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

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



244

245

**Figure 1 The SAML Domain Model**

246 One major design ~~contergoal~~ for SAML is Single Sign-On (SSO), the ability of a user to  
 247 authenticate in one domain and use resources in other domains without re-authenticating.  
 248 However, SAML can be used in various configurations to support additional scenarios as well.  
 249 Several profiles of SAML are defined that support different styles of SSO and the securing of SOAP  
 250 payloads.

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

### 255 **1.3.2. SAML and URI-Based Identifiers**

256 SAML defines some identifiers to manage references to well-known concepts and sets of values.  
257 For example, the SAML-defined identifier for the Kerberos subject confirmation method is as  
258 follows:

259 **urn:ietf:rfc:1510**

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

262 ~~[http://www.oasis-open.org/committees/security/docs/draft-sstc-core-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/rwcdc)~~  
263 ~~[25/rwcdc](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26/rwcdc)~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26/rwcdc>

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

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

274 See [section 7](#) for a list of SAML-defined identifiers.

### 275 **1.3.3. SAML and Extensibility**

276 The XML formats for SAML assertions and protocol messages have been designed to ~~support~~  
277 ~~extension.be extensible.~~

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

## 2. SAML Assertions

280

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

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

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

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

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

### 2.1. Schema Header and Namespace Declarations

292

293 The following schema fragment defines the XML namespaces and other header information for the  
294 assertion schema:

```
295 <schema  
296   targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-  
297   sstc-schema-assertion-26.xsd"  
298   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
299   xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-  
300   schema-assertion-26.xsd"  
301   xmlns="http://www.w3.org/2001/XMLSchema"  
302   elementFormDefault="unqualified">  
303   <import namespace="http://www.w3.org/2000/09/xmldsig#"  
304     schemaLocation="xmldsig-core-schema.xsd"/>  
305   <annotation>  
306     <documentation>draft-sstc-schema-assertion-26.xsd</documentation>  
307   </annotation>  
308   ...  
309 </schema>
```

### 2.2. Simple Types

310

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

#### ~~2.2.1. Simple Type IDType~~

312

~~313 The IDType simple type is used to declare and reference identifiers to assertions, requests, and  
314 responses.~~

~~315 Values of attributes declared to be of type IDType MUST satisfy the following properties:~~

~~316 ? Any party that assigns an identifier MUST ensure that there is negligible probability that that  
317 party or any other party will assign the same identifier to a different data object.~~

~~318 ? Where a data object declares that it has a particular identifier, there MUST be exactly one  
319 such declaration.~~

~~320 The mechanism by which the application ensures that the identifier is unique is left to the  
321 implementation. In the case that a pseudorandom technique is employed, the probability of two  
322 randomly chosen identifiers being identical MUST be less than  $2^{-128}$  and SHOULD be less than  
323  $2^{-160}$ .~~

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

### 327 **2.2.1. The following schema fragment defines the IDType simple type:Types** 328 **IDType and IDReferenceType**

329 The IDType simple type is used to declare identifiers to assertions, requests, and responses. The  
330 IDReferenceType is used to reference identifiers of type IDType.

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

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

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

336 The mechanism by which the application ensures that the identifier is unique is left to the  
337 implementation. In the case that a pseudorandom technique is employed, the probability of two  
338 randomly chosen identifiers being identical MUST be less than  $2^{-128}$  and SHOULD be less than  
339  $2^{-160}$ . This requirement MAY be met by applying Base64 encoding to a randomly chosen value 128  
340 or 160 bits in length.

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

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

```
345 <simpleType name="IDType">  
346 <restriction base="string"/>  
347 </simpleType>  
348 <simpleType name="IDReferenceType">  
349 <restriction base="string"/>  
350 </simpleType>
```

### 351 **2.2.2. Simple Type DecisionType**

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

354 Permit

355 The specified action is permitted.

356 Deny

357 The specified action is denied.

358 Indeterminate

359 No assessment is made as to whether the specified action is permitted or denied.

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

```
361 <simpleType name="DecisionType">  
362 <restriction base="string">  
363 <enumeration value="Permit"/>  
364 <enumeration value="Deny"/>  
365 <enumeration value="Indeterminate"/>  
366 </restriction>  
367 </simpleType>
```

## 368 2.3. Assertions

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

### 370 2.3.1. Element <AssertionSpecifier>

371 The <AssertionSpecifier> element specifies an assertion either by reference or by value. It  
372 contains one of the following elements:

373 | <AssertionIDReference>

374 | Specifies an assertion by reference to the value of the assertion's AssertionID attribute.

375 | <Assertion>

376 | Specifies an assertion by value.

377 The following schema fragment defines the <AssertionSpecifier> element and its  
378 **AssertionSpecifierType** complex type:

```
379 | <element name="AssertionSpecifier" type="saml:AssertionSpecifierType" />  
380 | <complexType name="AssertionSpecifierType">  
381 |   <choice>  
382 |     <element ref="saml:AssertionIDReference" />  
383 |     <element ref="saml:Assertion" />  
384 |   </choice>  
385 | </complexType>
```

### 386 2.3.2. Element <AssertionID>

387 | The <AssertionID> element makes a reference to a SAML assertion by means of the value of  
388 | the assertion's AssertionID attribute.

389 The following schema fragment defines the <AssertionID> element:

```
390 | <element name="AssertionID" type="saml:IDType" /> <name="AssertionIDReference"  
391 | type="saml:IDReferenceType" />
```

### 392 2.3.3. Element <Assertion>

393 The <Assertion> element is of **AssertionType** complex type. This type specifies the basic  
394 | information that is common to all assertions, including the following elements ~~(in order)~~ and  
395 | attributes:

396 | MajorVersion [Required]

397 | The major version of this assertion. The identifier for the version of SAML defined in this  
398 | specification is 1. Processing of this attribute is specified in Section [1.1.1](#).

399 | MinorVersion [Required]

400 | The minor version of this assertion. The identifier for the version of SAML defined in this  
401 | specification is 0. Processing of this attribute is specified in Section [1.1.1](#).

402 | AssertionID [Required]

403 | The identifier for this assertion. It is of type **IDType**, and MUST follow the requirements  
404 | specified by that type for identifier uniqueness.

405 | Issuer [Required]

406 | The issuer of the assertion. The name of the issuer is provided as a string. The issuer  
407 | name SHOULD be unambiguous to the intended relying parties. SAML applications may  
408 | use an identifier such as a URI that is designed to be unambiguous regardless of context.

409 IssueInstant [Required]  
 410 The time instant of ~~issue. It has the type `dateTime`, which is built in to the W3C XML~~  
 411 ~~Schema Datatypes specification [Schema2]~~~~issue in UTC as described in section 1.2.1.~~  
 412 <Conditions> [Optional]  
 413 Conditions that MUST be taken into account in assessing the validity of the assertion.  
 414 <Advice> [Optional]  
 415 Additional information related to the assertion that assists processing in certain situations  
 416 but which MAY be ignored by applications that do not support its use.

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

419 One or more of the following statement elements:

420 <Statement>  
 421 A statement defined in an extension schema.  
 422 <SubjectStatement>  
 423 A subject statement defined in an extension schema.  
 424 <AuthenticationStatement>  
 425 An authentication statement.  
 426 <AuthorizationDecisionStatement>  
 427 An authorization decision statement.  
 428 <AttributeStatement>  
 429 An attribute statement.

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

```

432 <element name="Assertion" type="saml:AssertionType" />
433 <complexType name="AssertionType">
434   <sequence>
435     <element ref="saml:Conditions" minOccurs="0" />
436     <element ref="saml:Advice" minOccurs="0" />
437     <choice minOccurs="0" maxOccurs="unbounded">
438       <element ref="saml:Statement" />
439       <element ref="saml:SubjectStatement" />
440       <element ref="saml:AuthenticationStatement" />
441       <element ref="saml:AuthorizationDecisionStatement" />
442       <element ref="saml:AttributeStatement" />
443     </choice>
444     <element ref="ds:Signature" minOccurs="0"
445 maxOccurs="unbounded" />ref="ds:Signature" minOccurs="0" />
446   </sequence>
447   <attribute name="MajorVersion" type="integer" use="required" />
448   <attribute name="MinorVersion" type="integer" use="required" />
449   <attribute name="AssertionID" type="saml:IDType" use="required" />
450   <attribute name="Issuer" type="string" use="required" />
451   <attribute name="IssueInstant" type="dateTime" use="required" />
452 </complexType>

```

### 453 2.3.3.1. Element <Conditions>

454 If an assertion contains a <Conditions> element, the validity of the assertion is dependent on the  
 455 conditions provided. Each condition evaluates to a status of Valid, Invalid, or  
 456 Indeterminate. The validity status of an assertion is the conjunction of the validity status of each  
 457 of the conditions it contains, as follows:

458 ?? If any condition evaluates to `Invalid`, the assertion status is `Invalid`.  
459 ?? If no condition evaluates to `Invalid` and one or more conditions evaluate to  
460 `Indeterminate`, the assertion status is `Indeterminate`.  
461 ?? If no conditions are supplied or all the specified conditions evaluate to `Valid`, the assertion  
462 status is `Valid`.

463 Note that an assertion that has validity status 'Valid' may not be trustworthy by reasons such as not  
464 being issued by a trustworthy issuer or not being authenticated by a trustworthy signature.

465 The `<Conditions>` element MAY be extended to contain additional conditions. If an element  
466 contained within a `<Conditions>` element is encountered that is not understood, the status of the  
467 condition MUST be evaluated to `Indeterminate`.

468 The `<Conditions>` element ~~contains~~MAY contain the following elements and attributes:

469 `NotBefore` [Optional]

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

472 `NotOnOrAfter` [Optional]

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

475 `<Condition>` [~~Zero or more~~

476 Any Number]

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

478 `<AudienceRestrictionCondition>` [Any Number]

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

480 `<TargetRestrictionCondition>` [Any Number]

481 The `<TargetRestriction>` condition is used to limit the use of the assertion to a particular  
482 relying party.

483 The following schema fragment defines the `<Conditions>` element and its **ConditionsType**  
484 complex type:

```
485 <element name="Conditions" type="saml:ConditionsType"/>
486 <complexType name="ConditionsType">
487   <choice minOccurs="0" maxOccurs="unbounded">
488     <element ref="saml:Condition"/>
489     <element ref="saml:AudienceRestrictionCondition"/>
490     <element ref="saml:TargetRestrictionCondition"/>
491   </choice>
492   <attribute name="NotBefore" type="dateTime" use="optional"/>
493   <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>
494 </complexType>
```

#### 495 **2.3.3.1.1 Attributes `NotBefore` and `NotOnOrAfter`**

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

497 The `NotBefore` attribute specifies the time instant at which the validity interval begins. The

498 `NotOnOrAfter` attribute specifies the time instant at which the validity interval has ended.

499 If the value for either `NotBefore` or `NotOnOrAfter` is omitted or is equal to the start of the epoch,  
500 it is considered unspecified. If the `NotBefore` attribute is unspecified (and if any other conditions  
501 that are supplied evaluate to `Valid`), the assertion is valid at any time before the time instant  
502 specified by the `NotOnOrAfter` attribute. If the `NotOnOrAfter` attribute is unspecified (and if any  
503 other conditions that are supplied evaluate to `Valid`), the assertion is valid from the time instant

504 specified by the `NotBefore` attribute with no expiry. If neither attribute is specified (and if any other  
505 conditions that are supplied evaluate to `Valid`), the assertion is valid at any time.

506 The `NotBefore` and `NotOnOrAfter` attributes are defined to have the **dateTime** simple type that  
507 is built in to the W3C XML Schema Datatypes specification [**Schema2**]. All time instants are  
508 interpreted to be in Universal Coordinated Time (UTC) unless they explicitly indicate a time zone as  
509 described in section 1.2.1. Implementations MUST NOT generate time instants that specify leap  
510 seconds.

### 511 2.3.3.1.2 Element `<Condition>`

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

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

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

### 519 2.3.3.1.3 Elements `<AudienceRestrictionCondition>` and `<Audience>`

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

525 `<Audience>`

526 An audience is identified by a URI-A URI that identifies an intended audience. The URI  
527 MAY identify a document that describes the terms and conditions of audience membership.

528 The ~~condition~~ `AudienceRestrictionCondition` evaluates to `Valid` if and only if the relying  
529 party is a member of one or more of the audiences specified.

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

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

```
538 <element name="AudienceRestrictionCondition"  
539     type="saml:AudienceRestrictionConditionType" />  
540 <complexType name="AudienceRestrictionConditionType">  
541     <complexContent>  
542         <extension base="saml:ConditionAbstractType">  
543             <sequence>  
544                 <element ref="saml:Audience"  
545                     minOccurs="1" maxOccurs="unbounded" />  
546             </sequence>  
547         </extension>  
548     </complexContent>  
549 </complexType>  
550 <element name="Audience" type="anyURI" />
```

551 **2.3.3.1.4 ~~Condition Type TargetRestrictionType Elements~~**  
552 **~~<TargetRestrictionCondition> and <Target>~~**

553 The <TargetRestrictionCondition> element is used to limit the use of the assertion to a particular  
554 relying party. This is useful to prevent malicious forwarding of assertions to unintended recipients. It  
555 contains the following elements:

556 <Target>  
557 A URI that identifies an intended relying party.

558 The ~~target is identified by a URI. The condition evaluates to true~~ TargetRestrictionCondition  
559 evaluates to valid if and only if one or more URIs identify the recipient or a resource managed by  
560 the recipient.

561 The following schema fragment defines the <TargetRestrictionCondition> element and its  
562 **TargetRestrictionConditionType** complex type:

```
563 <element name="TargetRestrictionCondition "  
564         type="saml:TargetRestrictionConditionType" />  
565 <complexType name="TargetRestrictionConditionType">  
566   <complexContent>  
567     <extension base="saml:ConditionAbstractType">  
568       <sequence>  
569         <element ref="saml:Target "  
570             minOccurs="1" maxOccurs="unbounded" />  
571       </sequence>  
572     </extension>  
573   </complexContent>  
574 </complexType>  
575 <element name="Target " type="anyURI" />
```

576 **2.3.3.2. Elements <Advice> and <AdviceElement>**

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

580 The <Advice> element contains a mixture of zero or more <AssertionSpecifier> elements,  
581 <AdviceElement> elements, and elements in other namespaces, with lax schema validation in  
582 effect for these other elements.

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

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

588 The following schema fragment defines the <Advice> element and its **AdviceType** complex type,  
589 along with the <AdviceElement> element and its **AdviceAbstractType** complex type:

```
590 <element name="Advice " type="saml:AdviceType" />  
591 <complexType name="AdviceType">  
592   <sequence>  
593     <choice minOccurs="0" maxOccurs="unbounded">  
594       <element ref="saml:AssertionSpecifier" />  
595       <element ref="saml:AdviceElement" />  
596       <any namespace="##other" processContents="lax" />  
597     </choice>  
598   </sequence>  
599 </complexType>  
600 <element name="AdviceElement " type="saml:AdviceAbstractType" />
```

601 `<complexType name="AdviceAbstractType"/>`

## 602 2.4. Statements

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

### 604 2.4.1. Element <Statement>

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

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

```
610 <element name="Statement" type="saml:StatementAbstractType"/>  
611 <complexType name="StatementAbstractType" abstract="true"/>
```

### 612 2.4.2. Element <SubjectStatement>

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

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

```
620 <element name="SubjectStatement" type="saml:SubjectStatementAbstractType"/>  
621 <complexType name="SubjectStatementAbstractType" abstract="true">  
622   <complexContent>  
623     <extension base="saml:StatementAbstractType">  
624       <sequence>  
625         <element ref="saml:Subject"/>  
626       </sequence>  
627     </extension>  
628   </complexContent>  
629 </complexType>
```

#### 630 2.4.2.1. Element <Subject>

631 The <Subject> element specifies one or more subjects. It contains either or both of the following  
632 elements:

633 <NameIdentifier>

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

635 <SubjectConfirmation>

636 Information that allows the subject to be authenticated.

637 If a <Subject> element contains more than one subject specification, the issuer is asserting that  
638 the surrounding statement is true for all of the subjects specified. For example, if both a  
639 <NameIdentifier> and a <SubjectConfirmation> element are present, the issuer is  
640 asserting that the statement is true of both subjects being identified. A <Subject> element  
641 SHOULD NOT identify more than one principal.

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

```
644 <element name="Subject" type="saml:SubjectType"/>  
645 <complexType name="SubjectType">
```

```

646 | <choice maxOccurs="unbounded">
647 |   <sequence>
648 |     <element ref="saml:NameIdentifier"/>
649 |     <element ref="saml:SubjectConfirmation" minOccurs="0"/>
650 |   </sequence>
651 |   <element ref="saml:SubjectConfirmation"/>
652 | </choice>
653 | </complexType>

```

#### 654 2.4.2.2. Element <NameIdentifier>

655 The <NameIdentifier> element specifies a subject by a combination of a name and a security  
656 domain. It has the following attributes:

657 SecurityDomain

658 The security domain governing the name of the subject.

659 Name

660 The name of the subject.

661 The interpretation of the security domain and the name are left to individual implementations,  
662 including issues of anonymity, pseudonymity, and the persistence of the identifier with respect to  
663 the asserting and relying parties.

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

```

666 | <element name="NameIdentifier" type="saml:NameIdentifierType"/>
667 | <complexType name="NameIdentifierType">
668 |   <attribute name="SecurityDomain" type="string"/>
669 |   <attribute name="Name" type="string"/>
670 | </complexType>

```

#### 671 2.4.2.3. Elements <SubjectConfirmation>, <ConfirmationMethod>, and 672 <SubjectConfirmationData>

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

675 <ConfirmationMethod> [One or more]

676 A URI that identifies a protocol to be used to authenticate the subject. URIs identifying  
677 common authentication protocols are listed in Section 7.

678 <SubjectConfirmationData> ~~[Zero or more]~~

679 [Optional]

680 Additional authentication information to be used by a specific authentication protocol.

681 <ds:KeyInfo> [Optional]

682 An XML Signature **[XMLSig]** element that specifies a cryptographic key held by the  
683 subject.

684 The following schema fragment defines the <SubjectConfirmation> element and its  
685 **SubjectConfirmationType** complex type, along with the <SubjectConfirmationData>  
686 element and the <ConfirmationMethod> element:

```

687 | <element name="SubjectConfirmation" type="saml:SubjectConfirmationType"/>
688 | <complexType name="SubjectConfirmationType">
689 |   <sequence>
690 |     <element ref="saml:ConfirmationMethod" maxOccurs="unbounded"/>
691 |     <element ref="saml:SubjectConfirmationData" minOccurs="0"/>
692 |     <element ref="ds:KeyInfo" minOccurs="0"/>
693 |   </sequence>
694 | </complexType>

```

```
695 <element name="SubjectConfirmationData" type="string" minOccurs="0"/> />
696 <element name="ConfirmationMethod" type="anyURI" />
```

### 697 2.4.3. Element <AuthenticationStatement>

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

702 AuthenticationMethod **[Required]**

703 **[Optional]**

704 A URI that specifies the type of authentication that took place. URIs identifying common  
705 authentication protocols are listed in Section 7.

706 AuthenticationInstant **[Required]**

707 **[Optional]**

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

710 <AuthenticationLocality> [Optional]

711 Specifies the DNS domain name and IP address for the system entity that from which the  
712 Subject was apparently authenticated.

713 <AuthenticationBinding> [Any Number]

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

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

```
717 <element name="AuthenticationStatement"
718         type="saml:AuthenticationStatementType" />
719 <complexType name="AuthenticationStatementType">
720   <complexContent>
721     <extension base="saml:SubjectStatementAbstractType">
722       <sequence>
723         <element ref="saml:AuthenticationLocality" minOccurs="0" />
724         <element ref="saml:AuthorityBinding"
725                 minOccurs="0" maxOccurs="unbounded" />
726       </sequence>
727       <attribute name="AuthenticationMethod" type="anyURI" />
728       <attribute name="AuthenticationInstant" type="dateTime" />
729     </extension>
730   </complexContent>
731 </complexType>
```

#### 732 2.4.3.1. Element <AuthenticationLocality>

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

735 IPAddress [Optional]

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

737 DNSAddress **[Required]**

738 **[Optional]**

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

740 This element is entirely advisory, since both these fields are quite easily "spoofed" but current  
741 practice appears to require its inclusion.

742 The following schema fragment defines the <AuthenticationLocality> element and its  
743 **AuthenticationLocalityType** complex type:

```
744 <element name="AuthenticationLocality"  
745         type="saml:AuthenticationLocalityType"/>  
746 <complexType name="AuthenticationLocalityType ">  
747   <attribute name="IPAddress" type="string" use="optional" />  
748   <attribute name="DNSAddress" type="string" use="optional" />  
749 </complexType>
```

### 750 2.4.3.2. Element <AuthorityBinding>

751 The <AuthorityBinding> element ~~specifies the type of authority (authentication, attribute,~~  
752 ~~may be used to indicate to a relying party receiving an AuthenticationStatement that a SAML~~  
753 ~~authority may be available to provide additional information about the subject of the statement. A~~  
754 ~~single SAML authority may advertise its presence over multiple protocol bindings, at multiple~~  
755 ~~locations, and as more than one kind of authorization) that performed the authentication and points~~  
756 ~~to it via URI:~~

757 ~~AuthorityKind [Optional]~~

758 ~~The type of authority that performed the authentication.~~

759 ~~Binding [Optional]~~

760 ~~The address of the authority authority by sending multiple elements as needed.~~

761 AuthorityKind [Required]

762 The type of SAML authority (Authentication, Attribute, or Authorization Decision) advertised  
763 by the element. The kind of authority corresponds to the derived type of SubjectQuery that  
764 the authority expects to receive (and is likely to be able to successfully answer) at the  
765 location being advertised. For example, a value of "attribute" means that an  
766 <AttributeQuery> is expected.

767 Location [Required]

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

771 Binding [Required]

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

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

```
776 <element name="AuthorityBinding" type="saml:AuthorityBindingType"/>  
777 <complexType name="AuthorityBindingType">  
778   <attribute name="AuthorityKind" type="saml:AuthorityKindType" />  
779   <attribute name="Location" type="anyURI" use="required" />  
780   <attribute name="Binding" type="anyURI" use="required" />  
781 </complexType>  
782 <simpleType name="AuthorityKindType">  
783   <restriction base="string">  
784     <enumeration value="authentication" />  
785     <enumeration value="attribute" />  
786     <enumeration value="authorization" />  
787   </restriction>  
788 </simpleType>
```

### 789 2.4.4. Element <AuthorizationDecisionStatement>

790 The <AuthorizationDecisionStatement> element supplies a statement by the issuer that the  
791 request for access by the specified subject to the specified resource has resulted in the specified

792 decision on the basis of some optionally specified evidence. It is of type  
793 **AuthorizationDecisionStatementType**, which extends **SubjectStatementAbstractType** with the  
794 addition of the following elements (in order) and attributes:

795 Resource [Optional]

796 A URI identifying the resource to which access authorization is sought.

797 Decision [Optional]

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

800 <Actions> [Required]

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

802 <Evidence> [~~Zero or more~~]

803 [~~Any Number~~]

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

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

```
807 <element name="AuthorizationDecisionStatement"
808 type="saml:AuthorizationDecisionStatementType" />
809 <complexType name="AuthorizationDecisionStatementType">
810 <complexContent>
811 <extension base="saml:SubjectStatementAbstractType">
812 <sequence>
813 <element ref="saml:Actions" />
814 <element ref="saml:Evidence" minOccurs="0"
815 maxOccurs="unbounded" />
816 </sequence>
817 <attribute name="Resource" type="anyURI" use="optional" />
818 <attribute name="Decision" type="saml:DecisionType"
819 use="optional" />
820 </extension>
821 </complexContent>
822 </complexType>
```

#### 823 2.4.4.1. Elements <Actions> and <Action>

824 The <Actions> element specifies the set of actions on the specified resource for which permission  
825 is sought. It has the following element and attribute:

826 Namespace [Optional]

827 A URI representing the namespace in which the names of specified actions are to be  
828 interpreted. If this element is absent, the namespace [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#rwedc-negation)  
829 [open.org/committees/security/docs/draft-sstc-core-26#rwedc-negation](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#rwedc-negation) specified in section  
830 7.2.2 is in effect ~~by default~~.

831 <Action> [One or more]

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

833 The following schema fragment defines the <Actions> element, its **ActionsType** complex type,  
834 and the <Action> element:

```
835 <element name="Actions" type="saml:ActionsType" />
836 <complexType name="ActionsType">
837 <sequence>
838 <element ref="saml:Action" maxOccurs="unbounded" />
839 </sequence>
840 <attribute name="Namespace" type="anyURI" use="optional" />
841 </complexType>
842 <element name="Action" type="string" />
```

#### 843 2.4.4.2. Element <Evidence>

844 The <Evidence> element contains an assertion that the issuer relied on in issuing the  
845 authorization decision. It has the **AssertionSpecifierType** complex type.

846 The provision of an assertion as evidence MAY affect the reliance agreement between the **client**  
847 ~~and the service-requestor and the Authorization Authority~~. For example, in the case that the  
848 **clientrequestor** presented an assertion to the **serviceAuthorization Authority** in a request, the  
849 **serviceAuthorization Authority** MAY use that assertion as evidence in making its response without  
850 endorsing the assertion as valid either to the **clientrequestor** or any third party.

851 The following schema fragment defines the <Evidence> element:

```
852 <element name="Evidence" type="saml:AssertionSpecifierType" />
```

#### 853 2.4.5. Element <AttributeStatement>

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

857 <Attribute> [One or More]

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

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

```
861 <element name="AttributeStatement" type="saml:AttributeStatementType" />  
862 <complexType name="AttributeStatementType">  
863 <complexContent>  
864 <extension base="saml:SubjectStatementAbstractType">  
865 <sequence>  
866 <element ref="saml:Attribute" maxOccurs="unbounded" />  
867 </sequence>  
868 </extension>  
869 </complexContent>  
870 </complexType>
```

#### 871 2.4.5.1. Elements <AttributeDesignator> and <Attribute>

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

876 AttributeNamespace **[Required]**

877 **[Optional]**

878 The namespace in which the AttributeName elements are interpreted.

879 AttributeName **[Required]**

880 **[Optional]**

881 The name of the attribute.

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

```
884 <element name="AttributeDesignator" type="saml:AttributeDesignatorType" />  
885 <complexType name="AttributeDesignatorType">  
886 <attribute name="AttributeName" type="string" />  
887 <attribute name="AttributeNamespace" type="anyURI" />  
888 </complexType>
```

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

892 <AttributeValue> ~~[Required]~~  
893 ~~[Any Number]~~  
894 The value of the attribute.

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

```
897 <element name="Attribute" type="saml:AttributeType" />  
898 <complexType name="AttributeType">  
899 <complexContent>  
900 <extension base="saml:AttributeDesignatorType">  
901 <sequence>  
902 <element ref="saml:AttributeValue" maxOccurs="unbounded" />  
903 </sequence>  
904 </extension>  
905 </complexContent>  
906 </complexType>
```

#### 907 2.4.5.1.1 Element <AttributeValue>

908 The <AttributeValue> element supplies the value of ~~the~~ specified attribute. It is of the  
909 ~~AttributeValueType~~ complex anyType simple type, which allows the inclusion of any element in  
910 any namespace and specifies that lax schema validation is in effect.any well-formed XML to appear  
911 as the content of the element.

912 If the data content of an AttributeValue element is of a XML Schema simple type (e.g. interger,  
913 string, etc) the data type MAY be declared explicitly by means of an `xsi:type` declaration in the  
914 <AttributeValue> element. If the attribute value contains structured data the necessary data  
915 elements may be defined in an extension schema introduced by means of the `xmlns=` mechanism.

916 The following schema fragment defines the <AttributeValue> element ~~and its~~  
917 **AttributeValueType** complex type:

```
918 <element name="AttributeValue"  
919 type="saml:AttributeValueType" />type="anyType" />  
920 <complexType name="AttributeValueType">  
921 <sequence>  
922 <any namespace="##any" processContents="lax"  
923 minOccurs="0" maxOccurs="unbounded" />  
924 </sequence>  
925 </complexType>
```

## 3. SAML Protocol

926

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

930 SAML-aware ~~client requestors~~ MAY in addition use the SAML request-response protocol defined  
931 by the <Request> and <Response> elements. The ~~client requestor~~ sends a <Request> element  
932 to a SAML ~~service, and the service authority, and the authority~~, and the authority generates a <Response> element,  
933 as shown in ~~Figure 2~~ Figure 2.



934

935

Figure 2: SAML Request-Response Protocol

### 3.1. Schema Header and Namespace Declarations

936

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

```
939 <schema  
940   targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-  
941   sstc-schema-protocol-26.xsd"  
942   xmlns="http://www.w3.org/2001/XMLSchema"  
943   xmlns:samlp="http://www.oasis-open.org/committees/security/docs/draft-sstc-  
944   schema-protocol-26.xsd"  
945   xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-  
946   schema-assertion-26.xsd"  
947   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
948   elementFormDefault="unqualified">  
949   <import namespace="http://www.oasis-open.org/committees/security/docs/draft-  
950   sstc-schema-assertion-26.xsd"  
951     schemaLocation="draft-sstc-schema-assertion-26.xsd" />  
952   <import namespace="http://www.w3.org/2000/09/xmldsig#"  
953     schemaLocation="xmldsig-core-schema.xsd" />  
954   <annotation>  
955     <documentation>draft-sstc-schema-protocol-26.xsd</documentation>  
956   </annotation>  
957   ...  
958 </schema>
```

959

### 3.2. Requests

960

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

#### 3.2.1. Complex Type RequestAbstractType

962

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

966 RequestID [Required]

967

An identifier for the request. It is of type **IDType**, and MUST follow the requirements

968 specified by that type for identifier uniqueness. The values of the `RequestID` attribute in a  
969 request and the `InResponseTo` attribute in the corresponding response MUST match.

970 `MajorVersion` [Required]

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

973 `MinorVersion` [Required]

974 The minor version of this request. The identifier for the version of SAML defined in this  
975 specification is 0. Processing of this attribute is specified in Section 3.4.2.

976 `IssueInstant` [Required]

977 The time instant of issue of the request. The time value is encoded in UTC as described in  
978 section 1.2.1.

979 `<RespondWith>` [Any Number]

980 Each `<RespondWith>` element specifies a type of response that is acceptable to the  
981 requestor.

982 `<Signature>` [Optional]

983 An XML Signature that authenticates the assertion, see section 5.

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

```
985 <complexType name="RequestAbstractType" abstract="true">  
986 <sequence>  
987 <element ref="samlp:RespondWith"  
988 minOccurs="0" maxOccurs="unbounded"/>  
989 <element ref="ds:Signature" minOccurs="0"  
990 maxOccurs="unbounded"/>"ds:Signature" minOccurs="0"/>  
991 </sequence>  
992 <attribute name="RequestID" type="saml:IDType" use="required"/>  
993 <attribute name="MajorVersion" type="integer" use="required"/>  
994 <attribute name="MinorVersion" type="integer" use="required"/>  
995 <attribute name="IssueInstant" type="dateTime" use="required"/>  
996 </complexType>
```

### 997 3.2.1.1. Element `<RespondWith>`

998 The `<RespondWith>` element specifies a type of response that is acceptable to the requestor. If  
999 no `<RespondWith>` element is specified the default is `SingleStatement`.

1000 `SingleStatement`. The `<RespondWith>` element specifies the type(s) of response that is  
1001 acceptable to the requestor. Multiple `<RespondWith>` elements MAY be specified to indicate that  
1002 the requestor is capable of processing multiple requests.

1003 `<RespondWith>` elements are used to inform the responder of the type of assertion statements  
1004 that the requestor is capable of processing. The Responder MUST use this information to ensure  
1005 that it generates responses consistent with information found in the `<RespondWith>` element of  
1006 the Request.

1007 NOTE: Inability to find assertions that meet `<RespondWith>` criteria should be treated identical to  
1008 any other query for which no assertions are available. In both cases a status of success would  
1009 normally be returned in the Response message, but no assertions to be found therein.

1010 Acceptable values for the `<RespondWith>` element are:

1011 `SingleStatement`

1012 An assertion carrying exactly one statement element.

1013 `MultipleStatement`

1014 An assertion carrying at least one statement element.

1015 AuthenticationStatement  
 1016 An assertion carrying an Authentication statement.

1017 AuthorizationDecisionStatement  
 1018 An assertion carrying an Authorization Decision statement.

1019 AttributeStatement  
 1020 An assertion carrying an Attribute statement.

1021 Schema URI  
 1022 An assertion containing additional elements from the specified schema.

1023 The following schema fragment defines the <RespondWith> element:  
 1024 `<element name="RespondWith" type="anyURI" />`

### 1025 3.2.2. Element <Request>

1026 The <Request> element specifies a SAML request. It provides either a query or a request for a  
 1027 specific assertion identified by <AssertionIDReference> or <AssertionArtifact>. It has  
 1028 the complex type **RequestType**, which extends **RequestAbstractType** by adding a choice of one  
 1029 of the following elements:

1030 <Query>  
 1031 An extension point that allows extension schemas to define new types of query.

1032 <SubjectQuery>  
 1033 An extension point that allows extension schemas to define new types of query that specify  
 1034 a single SAML subject.

1035 <AuthenticationQuery>  
 1036 Makes a query for authentication information.

1037 <AttributeQuery>  
 1038 Makes a query for attribute information.

1039 <AuthorizationDecisionQuery>  
 1040 Makes a query for an authorization decision.

1041 <AssertionIDReference> [One or more]  
 1042 Requests an assertion by reference to its assertion identifier.

1043 <AssertionArtifact> [One or more]  
 1044 Requests an assertion by supplying an assertion artifact that represents it.

1045 The following schema fragment defines the <Request> element and its **RequestType** complex  
 1046 type:

```
1047 <element name="Request" type="saml:RequestType"/>
1048 <complexType name="RequestType">
1049   <complexContent>
1050     <extension base="saml:RequestAbstractType">
1051       <choice>
1052         <element ref="saml:Query"/>
1053         <element ref="saml:SubjectQuery"/>
1054         <element ref="saml:AuthenticationQuery"/>
1055         <element ref="saml:AttributeQuery"/>
1056         <element ref="saml:AuthorizationDecisionQuery"/>
1057         <element ref="saml:AssertionIDReference" maxOccurs="unbounded"/>
1058         <element ref="saml:AssertionArtifact" maxOccurs="unbounded"/>
1059       </choice>
1060     </extension>
1061   </complexContent>
1062 </complexType>
```

### 1063 **3.2.3. Element <AssertionArtifact>**

1064 The <AssertionArtifact> element is used to specify the assertion artifact that represents an  
1065 assertion.

1066 The following schema fragment defines the <AssertionArtifact> element:

```
1067 <element name="AssertionArtifact" type="string"/>
```

## 1068 **3.3. Queries**

1069 The following sections define the SAML constructs that contain query information.

### 1070 **3.3.1. Element <Query>**

1071 The <Query> element is an extension point that allows new SAML queries to be defined. Its  
1072 **QueryAbstractType** is abstract; extension elements MUST use the `xsi:type` attribute to indicate  
1073 the derived type. **QueryAbstractType** is the base type from which all SAML query elements are  
1074 derived.

1075 The following schema fragment defines the <Query> element and its **QueryAbstractType**  
1076 complex type:

```
1077 <element name="Query" type="saml:QueryAbstractType"/>  
1078 <complexType name="QueryAbstractType" abstract="true"/>
```

### 1079 **3.3.2. Element <SubjectQuery>**

1080 The <SubjectQuery> element is an extension point that allows new SAML queries that specify a  
1081 single SAML subject. Its **SubjectQueryAbstractType** complex type is abstract; extension elements  
1082 MUST use the `xsi:type` attribute to indicate the derived type. **SubjectQueryAbstractType** adds  
1083 the <Subject> element.

1084 The following schema fragment defines the <SubjectQuery> element and its  
1085 **SubjectQueryAbstractType** complex type:

```
1086 <element name="SubjectQuery" type="saml:SubjectQueryAbstractType"/>  
1087 <complexType name="SubjectQueryAbstractType" abstract="true">  
1088 <complexContent>  
1089 <extension base="saml:QueryAbstractType">  
1090 <sequence>  
1091 <element ref="saml:Subject"/>  
1092 </sequence>  
1093 </extension>  
1094 </complexContent>  
1095 </complexType>
```

### 1096 **3.3.3. Element <AuthenticationQuery>**

1097 The <AuthenticationQuery> element is used to make the query “What authentication  
1098 assertions are available for this subject?” A successful response will be in the form of ~~an~~  
1099 ~~assertion~~ assertions containing ~~an~~ authentication statement s. This element is of type  
1100 **AuthenticationQueryType**, which extends **SubjectQueryAbstractType** with the addition of the  
1101 following element:

1102 <ConfirmationMethod> [Optional]

1103 A filter for possible responses. If it is present, the query made is “What authentication  
1104 assertions do you have for this subject with the supplied confirmation method?”

1105 In response to an authentication query, a responder returns assertions with authentication  
1106 statements as follows: The <Subject> element in the returned assertions MUST be identical to  
1107 the <Subject> element of the query. If the <ConfirmationMethod> element is present in the  
1108 query, at least one <ConfirmationMethod> element in the response MUST match. It is  
1109 OPTIONAL for the complete set of all such matching assertions to be returned in the response.

1110 The following schema fragment defines the <AuthenticationQuery> type and its  
1111 **AuthenticationQueryType** complex type:

```
1112 <element name="AuthenticationQuery" type="samlp:AuthenticationQueryType" />  
1113 <complexType name="AuthenticationQueryType">  
1114 <complexContent>  
1115 <extension base="samlp:SubjectQueryAbstractType">  
1116 <sequence>  
1117 <element ref="saml:ConfirmationMethod" minOccurs="0" />  
1118 </sequence>  
1119 </extension>  
1120 </complexContent>  
1121 </complexType>
```

### 1122 3.3.4. Element <AttributeQuery>

1123 The <AttributeQuery> element is used to make the query "Return the requested attributes for  
1124 this subject." **TheA successful** response will be in the form of **an assertion containing an attribute**  
1125 **statement-assertions containing attribute statements**. This element is of type **AttributeQueryType**,  
1126 which extends **SubjectQueryAbstractType** with the addition of the following element and attribute:

1127 <AttributeDesignator> ~~[Zero or more]~~**[Any Number]** (see Section 2.4.5.1)

1128 Each <AttributeDesignator> element specifies an attribute whose value is to be  
1129 returned. If no attributes are specified, the list of desired attributes is implicit and  
1130 application-specific.

1131 The following schema fragment defines the <AttributeQuery> element and its  
1132 **AttributeQueryType** complex type:

```
1133 <element name="AttributeQuery" type="samlp:AttributeQueryType" />  
1134 <complexType name="AttributeQueryType">  
1135 <complexContent>  
1136 <extension base="samlp:SubjectQueryAbstractType">  
1137 <sequence>  
1138 <element ref="saml:AttributeDesignator"  
1139 minOccurs="0" maxOccurs="unbounded" />  
1140 </sequence>  
1141 </extension>  
1142 </complexContent>  
1143 </complexType>
```

### 1144 3.3.5. Element <AuthorizationDecisionQuery>

1145 The <AuthorizationDecisionQuery> element is used to make the query "Should these  
1146 actions on this resource be allowed for this subject, given this evidence?" **TheA successful**  
1147 response will be in the form of **an assertion-assertions** containing **an** authorization decision  
1148 **statement<sub>s</sub>**. This element is of type **AuthorizationDecisionQueryType**, which extends  
1149 **SubjectQueryAbstractType** with the addition of the following elements and attribute:

1150 Resource [Required]

1151 A URI indicating the resource for which authorization is requested.

1152 <Actions> [Required]

1153 The actions for which authorization is requested.

1154 <Evidence> [\[Zero or more\]](#)  
1155 [\[Any Number\]](#)  
1156 An assertion that the responder MAY rely on in making its response.

1157 The following schema fragment defines the <AuthorizationDecisionQuery> element and its  
1158 **AuthorizationDecisionQueryType** complex type:

```
1159 <element name="AuthorizationDecisionQuery "  
1160 type="samlp:AuthorizationDecisionQueryType"/>  
1161 <complexType name="AuthorizationDecisionQueryType">  
1162 <complexContent>  
1163 <extension base="samlp:SubjectQueryAbstractType">  
1164 <sequence>  
1165 <element ref="saml:Actions"/>  
1166 <element ref="saml:Evidence"  
1167 minOccurs="0" maxOccurs="unbounded"/>  
1168 </sequence>  
1169 <attribute name="Resource" type="anyURI" use="required"/>  
1170 </extension>  
1171 </complexContent>  
1172 </complexType>
```

## 1173 3.4. Responses

1174 The following sections define the SAML constructs that contain response information.

### 1175 3.4.1. Complex Type ResponseAbstractType

1176 All SAML responses are of types that are derived from the abstract **ResponseAbstractType**  
1177 complex type. This type defines common attributes [and elements](#) that are associated with all SAML  
1178 responses:

1179 **ResponseID** [Required]

1180 An identifier for the response. It is of type **IDType**, and MUST follow the requirements  
1181 specified by that type for identifier uniqueness.

1182 **InResponseTo** [Required]

1183 A reference to the identifier of the request to which the response corresponds. The value of  
1184 this attribute MUST match the value of the corresponding **RequestID** attribute.

1185 **MajorVersion** [Required]

1186 The major version of this response. The identifier for the version of SAML defined in this  
1187 specification is 1. Processing of this attribute is specified in Section [1.1.1](#).

1188 **MinorVersion** [Required]

1189 The minor version of this response. The identifier for the version of SAML defined in this  
1190 specification is 0. Processing of this attribute is specified in Section [1.1.1](#).

1191 **IssueInstant** [Optional]

1192 [The time instant of issue of the request. The time value is encoded in UTC as described in](#)  
1193 [section 1.2.1.](#)

1194 **<Signature>** [Any Number]

1195 [An XML Signature that authenticates the assertion, see section 5.](#)

1196 The following schema fragment defines the **ResponseAbstractType** complex type:

```
1197 <complexType name="ResponseAbstractType" abstract="true" >  
1198 <sequence>  
1199 <element ref = "ds:Signature minOccurs="0"  
1200 maxOccurs="unbounded"/> "ds:Signature minOccurs="0"/>  
1201 </sequence>
```

```

1202     <attribute name="ResponseID" type="saml:IDType" use="required" />
1203     <attribute name="InResponseTo" type="saml:IDType" use="required" />
1204     type="saml:IDReferenceType"
1205     use="required" />
1206     <attribute name="MajorVersion" type="integer" use="required" />
1207     <attribute name="MinorVersion" type="integer" use="required" />
1208     <attribute name="IssueInstant" type="dateTime" use="required" />
1209 </complexType>

```

### 1210 3.4.2. Element <Response>

1211 The <Response> element specifies the status of the corresponding SAML request and a list of  
1212 zero or more assertions that answer the request. It has the complex type **ResponseType**, which  
1213 extends **ResponseAbstractType** by adding the following elements (in an unbounded mixture) ~~and~~  
1214 **attribute**:

1215 <Status> [Required] (see Section 3.4.3)  
1216 A code representing the status of the corresponding request.

1217 <Assertion> [Any Number] (see Section 2.3.3)  
1218 Specifies an assertion by value.

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

```

1221 <element name="Response" type="samlp:ResponseType" />
1222 <complexType name="ResponseType">
1223   <complexContent>
1224     <extension base="samlp:ResponseAbstractType">
1225       <sequence>
1226         <element ref="samlp:Status" />
1227         <element ref="saml:Assertion"
1228           minOccurs="0" maxOccurs="unbounded" />
1229       </sequence>
1230     </extension>
1231   </complexContent>
1232 </complexType>

```

### 1233 3.4.3. Element <Status>

1234 The <Status> element :

1235 <StatusCode> [Required]  
1236 A code representing the status of the corresponding request.

1237 <StatusMessage> [Any Number]  
1238 A message which MAY be returned to an operator.

1239 <StatusDetail> [Optional]  
1240 Specifies additional information concerning an error condition.

1241 The following schema fragment defines the <Status> element and its **StatusType** complex type:

```

1242 <element name="Status" type="samlp:StatusType" />
1243 <complexType name="StatusType">
1244   <sequence>
1245     <element ref="samlp:StatusCode" />
1246     <element ref="samlp:StatusMessage"
1247       minOccurs="0" maxOccurs="unbounded" />
1248     <element ref="samlp:StatusDetail" minOccurs="0" />
1249   </sequence>
1250 </complexType>

```

### 1251 3.4.3.1. Element <StatusCode>

1252 The <StatusCode> element specifies a code representing the status of the corresponding request  
1253 and an option sub code providing more specific information concerning a particular error status:

1254 Value [Required]

1255 The status code value as defined below.

1256 <SubStatusCode> [Optional]

1257 An optional subordinate status code value that provides more specific information on an  
1258 error condition.

1259 The following **StatusCode** values are defined:

1260 Success

1261 The request succeeded.

1262 VersionMismatch

1263 The receiver could not process the request because the version was incorrect.

1264 ~~Receiver~~

1265 Receiver

1266 The request could not be performed due to an error at the receiving end.

1267 Sender

1268 The request could not be performed due to an error in the sender or in the request

1269 The following schema fragment defines the <StatusCode> element and its **StatusCodeType**  
1270 complex type and the **StatusCodeEnumType** simple type:

```
1271 <element name="StatusCode" type="samlp:StatusCodeType"/>
1272 <complexType name="StatusCodeType">
1273   <sequence>
1274     <element ref="samlp:SubStatusCode" minOccurs="0"/>
1275   </sequence>
1276   <attribute name="Value" type="samlp:StatusCodeEnumType" use="required"/>
1277 </complexType>
1278 <simpleType name="StatusCodeEnumType">
1279   <restriction base="QName">
1280     <enumeration value="samlp:Success"/>
1281     <enumeration value="samlp:VersionMismatch"/>
1282     <enumeration value="samlp:Receiver"/>
1283     <enumeration value="samlp:Sender"/>
1284   </restriction>
1285 </simpleType>
```

### 1286 3.4.3.2. Element <SubStatusCode>

1287 The <SubStatusCode> element specifies an additional code representing the status of the  
1288 corresponding request:

1289 Value [Required]

1290 The status code value as defined below.

1291 <SubStatusCode> [Optional]

1292 An optional subordinate status code value that provides an additional level of specific  
1293 information on an error condition.

1294 The following **SubStatusCode** values are defined, additional codes MAY be defined in future  
1295 versions of the SAML specification:

1296 RequestVersionTooHigh

1297 The protocol version specified in the request is a major upgrade from the highest protocol  
1298 version supported by the responder.

- 1299 RequestVersionTooLow  
 1300       The responder cannot respond to the particular request using the SAML version specified  
 1301       in the request because it is too low.
- 1302 RequestVersionDeprecated  
 1303       The responder does not respond to any requests with the protocol version specified in the  
 1304       request.
- 1305 TooManyResponses  
 1306       The response would contain more elements than the responder will return.

1307 The following schema fragment defines the <SubStatusCode> element and its  
 1308 **SubStatusCodeType** complex type:

```
1309     <element name="SubStatusCode" type="samlp:SubStatusCodeType" />
1310     <complexType name="SubStatusCodeType">
1311       <sequence>
1312         <element ref="samlp:SubStatusCode" minOccurs="0" />
1313       </sequence>
1314       <attribute name="Value" type="QName" use="required" />
1315     </complexType>
```

### 1316 3.4.3.3. Element <StatusMessage>

1317 The <StatusMessage> element specifies a message that MAY be returned to an operator:

1318 The following schema fragment defines the <StatusMessage> element and its  
 1319 **StatusMessageType** complex type:

```
1320     <element name="StatusMessage" type="string" />
```

### 1321 3.4.3.4. Element <StatusDetail>

1322 The <StatusDetail> element MAY be used to specify additional information concerning an error  
 1323 condition.

1324 The following schema fragment defines the <StatusDetail> element and its **StatusDetailType**  
 1325 complex type:

```
1326     <element name="StatusDetail" type="samlp:StatusDetailType" />
1327     <complexType name="StatusDetailType">
1328       <sequence>
1329         <any namespace="##any"
1330             processContents="lax" minOccurs="0" maxOccurs="unbounded" />
1331       </sequence>
1332     </complexType>
```

## 1333 ~~3.4.4. Simple-Type StatusCodeType~~

1334 ~~The **StatusCodeType** simple type is used in a response to specify the status of the request that~~  
 1335 ~~caused the response to be generated. The type enumerates the following possible values:~~

1336 ~~Success~~

1337       ~~The request succeeded.~~

1338 ~~Failure~~

1339       ~~The request could not be performed by the service.~~

1340 ~~Error~~

1341       ~~An error in the request prevented the service from processing it.~~

1342 ~~Unknown~~

1343       ~~The request failed for unknown reasons.~~

1344 ~~The following schema fragment defines the **StatusCodeType** simple type:~~

```
1345 <simpleType name="StatusCodeType">  
1346 <restriction base="string">  
1347 <enumeration value="Success"/>  
1348 <enumeration value="Failure"/>  
1349 <enumeration value="Error"/>  
1350 <enumeration value="Unknown"/>  
1351 </restriction>
```

#### 1352 **3.4.4. </simpleType> Responses to <AuthenticationQuery> and** 1353 **<AttributeQuery>**

1354 Responses to Authentication and Attribute queries are constructed by matching against the  
1355 <saml:Subject> element found within the <AuthenticationQuery> or <AttributeQuery>  
1356 elements. In response to these queries, every assertion returned by a SAML responder MUST  
1357 contain at least one statement whose <saml:Subject> element **strongly matches** the  
1358 <saml:Subject> element found in the query.

1359 A <saml:Subject> element S1 strongly matches S2 if and only if:

- 1360 1 If S2 includes a <saml:NameIdentifier> element, then S1 must include an identical  
1361 <saml:NameIdentifier> element.
- 1362 2 If S2 includes a <saml:SubjectConfirmation> element, then S1 must include an  
1363 identical <saml:SubjectConfirmation> element.

## 4. SAML Versioning

1364

1365 SAML version information appears in the following elements:

1366 ?? <Assertion>

1367 ?? <Request>

1368 ?? <Response>

1369 The version numbering of the SAML assertion is independent of the version number of the SAML  
1370 request-response protocol. The version information for each consists of a major version number  
1371 and a minor version number, both of which are integers. In accordance with industry practice a  
1372 version number SHOULD be presented to the user in the form *Major.Minor*. This document defines  
1373 SAML Assertions 1.0 and SAML Protocol 1.0.

1374 The version number  $Major_B.Minor_B$  is higher than the version number  $Major_A.Minor_A$  if and only if:

1375  $Major_B > Major_A ? ( ( Major_B = Major_A ) ? Minor_B \Rightarrow Minor_A )$

1376 Each revision of SAML SHALL assign version numbers to assertions, requests, and responses that  
1377 are the same as or higher than the corresponding version number in the SAML version that  
1378 immediately preceded it.

1379 New versions of SAML SHALL assign new version numbers as follows:

1380 ?? **Documentation change:**  $( Major_B = Major_A ) ? ( Minor_B \Rightarrow Minor_A )$

1381 If the major and minor version numbers are unchanged, the new version *B* only introduces  
1382 changes to the documentation that raise no compatibility issues with an implementation of  
1383 version *A*.

1384 ?? **Minor upgrade:**  $( Major_B = Major_A ) ? ( Minor_B > Minor_A )$

1385 If the major version number of versions *A* and *B* are the same and the minor version  
1386 number of *B* is higher than that of *A*, the new SAML version MAY introduce changes to the  
1387 SAML schema and semantics but any changes that are introduced in *B* SHALL be  
1388 compatible with version *A*.

1389 ?? **Major upgrade:**  $Major_B > Major_A$

1390 If the major version of *B* number is higher than the major version of *A*, Version *B* MAY  
1391 introduce changes to the SAML schema and semantics that are incompatible with *A*.

### 4.1. Assertion Version

1392

1393 A SAML application MUST NOT issue any assertion whose version number is not supported.

1394 A SAML application MUST reject any assertion whose major version number is not supported.

1395 A SAML application MAY reject any assertion whose version number is higher than the highest  
1396 supported version.

### 4.2. Request Version

1397

1398 A SAML application SHOULD issue requests that specify the highest SAML version supported by  
1399 both the sender and recipient.

1400 If the SAML application does not know the capabilities of the recipient it should assume that it  
1401 supports the highest SAML version supported by the sender.

1402 **4.3. Response Version**

1403 A SAML application MUST NOT issue responses that specify a higher SAML version number than  
1404 the corresponding request.

1405 A SAML application MUST NOT issue a response that has a major version number that is lower  
1406 than the major version number of the corresponding request except to report the error  
1407 `RequestVersionTooHigh`.

1408 Incompatible protocol versions MAY cause the following errors to be reported:

1409 `RequestVersionTooHigh`

1410       The protocol version specified in the request is a major upgrade from the highest protocol  
1411       version supported by the responder.

1412 `RequestVersionTooLow`

1413       The responder cannot respond to the particular request using the SAML version specified  
1414       in the request because it is too low.

1415 `RequestVersionDeprecated`

1416       The responder does not respond to any requests with the protocol version specified in the  
1417       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 issuer (AP). This supports :

- (1) Message integrity
- (2) Authentication of the issuer to a relying party
- (3) If the signature is based on the issuer'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.

~~2~~Note :

?? SAML documents may be the subject of signatures from ~~in many~~ different packaging contexts. ~~[SIC];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) ~~thru~~through a secure channel and the AP has authenticated to the RP.

Request/Response messages:

~~the~~The originator has authenticated to the destination and the destination has obtained the assertion directly from the originator (no intermediaries) ~~thru~~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 issuer.
- (2) A SAML message ~~obtained~~ 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 ~~2.3.3-2.3~~.

## 1460 5.2. Request/Response Signing

1461 All SAML requests and responses MAY be signed using the XML Signature. This is reflected in the  
1462 schema – Section ~~3.3.1 & 3.5.1-3.2 & 3.4.~~

## 1463 5.3. Signature Inheritance ~~(a.k.a. super-signatures & sub-~~ 1464 ~~messages)~~

### 1465 5.3.1. Rationale

1466 SAML assertions may be embedded within request or response messages or other XML  
1467 messages, which may be signed. Request or response messages may themselves be contained  
1468 within other messages that are based on other XML messaging frameworks (e.g., SOAP) and the  
1469 composite object may be the subject of a signature. Another possibility is that SAML assertions or  
1470 request/response messages are embedded within a non-XML messaging object (e.g., MIME  
1471 package) and signed.

1472 In such a case, the SAML sub-message (Assertion, request, response) may be viewed as inheriting  
1473 a signature from the "super-signature" over the enclosing object, provided certain constraints are  
1474 met.

1475 (1) An assertion may be viewed as inheriting a signature from a super signature, if the super  
1476 signature applies all the elements within the assertion.

1477 A SAML request or response may be viewed as inheriting a signature from a super signature, if the  
1478 super signature applies to all ~~of~~ the elements within the response.

### 1479 5.3.2. Rules for SAML Signature Inheritance

1480 Signature inheritance occurs when SAML message (assertion/request/response) is not signed but  
1481 is enclosed within signed SAML such that the signature applies to all of the elements within the  
1482 message. In such a case, the SAML message is said to inherit the signature and may be  
1483 considered equivalent to the case where it is explicitly signed. The SAML message inherits the  
1484 "closest enclosing signature".

1485 But if SAML messages need to be passed around by themselves, or embedded in other messages,  
1486 they would need to be signed as per section ~~2-45.1~~

## 1487 5.4. XML Signature Profile

1488 The XML Signature [XMLSig] specification calls out a general XML syntax for signing data with  
1489 many flexibilities and choices. This section details the constraints on these facilities so that SAML  
1490 processors do not have to deal with the full generality of XML Signature processing.

### 1491 5.4.1. Signing formats

1492 XML Signature has three ways of representing signature in a document viz: enveloping, enveloped  
1493 and detached.

1494 SAML assertions and protocols MUST use the enveloped signatures for signing ~~assertions-~~  
1495 ~~assertions and protocols. SAML processors should support use of RSA signing and verification for~~  
1496 ~~public key operations.~~

1497 **5.4.2. CanonicalizationMethod**

1498 XML Signature REQUIRES the Canonical XML (omits comments)  
1499 (<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>). SAML implementations SHOULD use  
1500 Canonical XML with no comments.

1501 **5.4.3. Transforms**

1502 | ~~[Sig]~~**[XMLSig]** REQUIRES the enveloped signature transform  
1503 <http://www.w3.org/2000/09/xmlsig#enveloped-signature>

1504 **5.4.4. KeyInfo**

1505 | SAML does not restrict or impose any restrictions in this area. Therefore following ~~[Sig]~~**[XMLSig]**  
1506 keyInfo may be absent.

1507 **5.4.5. Binding between statements in a multi-statement assertion**

1508 Use of signing does not affect semantics of statements within assertions in any way, as stated in  
1509 this document Sections 1 ~~thru~~through 4.

1510 ~~**5.4.6. Security considerations**~~

1511 ~~**5.4.6.1. Replay Attack**~~

1512 ~~The mechanisms stated here in does not offer any counter measures against a replay attack. Other~~  
1513 ~~mechanisms like sequence numbers, time stamps, expiration et al need to be explored to prevent a~~  
1514 ~~replay attack.~~

1515

## 6. SAML Extensions

1516 The SAML schemas support extensibility. An example of an application that extends SAML  
1517 assertions is the XTAML system for management of embedded trust roots [XTAML]. The following  
1518 sections explain how to use the extensibility features in SAML to create extension schemas.

1519 Note that elements in the SAML schemas are not blocked from substitution, so that all SAML  
1520 elements MAY serve as the head element of a substitution group. Also, types are not defined as  
1521 *final*, so that all SAML types MAY be extended and restricted. The following sections discuss  
1522 only elements that have been specifically designed to support extensibility.

### 6.1. Assertion Schema Extension

1524 The SAML assertion schema is designed to permit separate processing of the assertion package  
1525 and the statements it contains, if the extension mechanism is used for either part.

1526 The following elements are intended specifically for use as extension points in an extension  
1527 schema; their types are set to *abstract*, so that the use of an *xsi:type* attribute with these  
1528 elements is REQUIRED:

1529 ?? <Assertion>

1530 ?? <Condition>

1531 ?? <Statement>

1532 ?? <SubjectStatement>

1533 ?? <AdviceElement>

1534 In addition, the following elements that are directly usable as part of SAML MAY be extended:

1535 ?? <AuthenticationStatement>

1536 ?? <AuthorizationDecisionStatement>

1537 ?? <AttributeStatement>

1538 ?? <AudienceRestrictionCondition>

1539 Finally, the following elements are defined to allow elements from arbitrary namespaces within  
1540 them, which serves as a built-in extension point without requiring an extension schema:

1541 ?? <AttributeValue>

1542 ?? <Advice>

### 6.2. Protocol Schema Extension

1544 The following elements are intended specifically for use as extension points in an extension  
1545 schema; their types are set to *abstract*, so that the use of an *xsi:type* attribute with these  
1546 elements is REQUIRED:

1547 ?? <Query>

1548 ?? <SubjectQuery>

1549 In addition, the following elements that are directly usable as part of SAML MAY be extended:

1550 ?? <Request>

1551        ?? <AuthenticationQuery>  
1552        ?? <AuthorizationDecisionQuery>  
1553        ?? <AttributeQuery>  
1554        ?? <Response>

### 1555        **6.3. Use of Type Derivation and Substitution Groups**

1556        W3C XML Schema [Schema1] provides two principal mechanisms for specifying an element of an  
1557        extended type: type derivation and substitution groups.

1558        For example, a <Statement> element can be assigned the type **NewStatementType** by means of  
1559        the `xsi:type` attribute. For such an element to be schema-valid, **NewStatementType** needs to be  
1560        derived from **StatementType**. The following example of a SAML assertion assumes that the  
1561        extension schema (represented by the `new:` prefix) has defined this new type:

```
1562        <saml:Assertion ...>  
1563            <saml:Statement xsi:type="new:NewStatementType">  
1564            ...  
1565            </saml:Statement>  
1566        </saml:Assertion>
```

1567        Alternatively, the extension schema can define a <NewStatement> element that is a member of a  
1568        substitution group that has <Statement> as a head element. For the substituted element to be  
1569        schema-valid, it needs to have a type that matches or is derived from the head element's type. The  
1570        following is an example of an extension schema fragment that defines this new element:

```
1571        <xsd:element "NewStatement" type="new:NewStatementType"  
1572            substitutionGroup="saml:Statement" />
```

1573        The substitution group declaration allows the <NewStatement> element to be used anywhere the  
1574        SAML <Statement> element can be used. The following is an example of a SAML assertion that  
1575        uses the extension element:

```
1576        <saml:Assertion ...>  
1577            <new:NewStatement>  
1578            ...  
1579            </new:NewStatement>  
1580        </saml:Assertion>
```

1581        The choice of extension method has no effect on the semantics of the XML document but does  
1582        have implications for interoperability.

1583        The advantages of type derivation are as follows:

1584        ?? A document can be more fully interpreted by a parser that does not have access to the  
1585        extension schema because a "native" SAML element is available.

1586        ?? At the time of writing, some W3C XML Schema validators do not support substitution  
1587        groups, whereas the `xsi:type` attribute is widely supported.

1588        The advantage of substitution groups is that a document can be explained without the need to  
1589        explain the functioning of the `xsi:type` attribute.

## 1590 7. SAML-Defined Identifiers

1591 The following sections define URI-based identifiers for common authentication protocols and  
1592 actions.

1593 Where possible an existing URN is used to specify a protocol. In the case of IETF protocols the  
1594 URN of the most current RFC that specifies the protocol is used. URIs created specifically for  
1595 SAML have the initial stem:

1596 <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/26>

### 1597 7.1. Confirmation Method Identifiers

1598 The following identifiers MAY be used in the <ConfirmationMethod> element (see Section  
1599 2.4.2.3) to refer to common authentication protocols.

#### 1600 7.1.1. SAML Artifact:

1601 **URI:** ~~[http://www.oasis-open.org/committees/security/docs/draft-sstc-core-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/artifact)~~  
1602 ~~25/artifact~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#artifact>

1603 <SubjectConfirmationData>: *Base64 ( Artifact )*

1604 The subject of the assertion is the party that can present the SAML Artifact value specified in  
1605 <SubjectConfirmationData>.

#### 1606 7.1.2. SAML Artifact (SHA-1):

1607 **URI:** ~~[http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/artifact-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/artifact-sha1)~~  
1608 ~~sha1~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#artifact-sha1>

1609 <SubjectConfirmationData>: *Base64 ( SHA1 ( Artifact ) )*

1610 The subject of the assertion is the party that can present a SAML Artifact such that the SHA1 digest  
1611 of the specified artifact matches the value specified in <SubjectConfirmationData>.

#### 1612 7.1.3. Holder of Key:

1613 **URI:** ~~[http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/Holder-Of-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/Holder-Of-Key)~~  
1614 ~~Key~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#Holder-Of-Key>

1615 <ds:KeyInfo>: Any cryptographic key

1616 The subject of the assertion is the party that can demonstrate that it is the holder of the private  
1617 component of the key specified in <ds:KeyInfo>.

#### 1618 7.1.4. Sender Vouches:

1619 **URI:** ~~[http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/sender-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/sender-vouches)~~  
1620 ~~vouches~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#sender-vouches>

1621 Indicates that no other information is available about the context of use of the assertion. The  
1622 Relying party SHOULD utilize other means to determine if it should process the assertion further.

1623 **7.1.5. Password (Pass-Through):**

1624 **URI:** ~~[http://www.oasis-open.org/committees/security/docs/draft-sstc-core-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/password)~~  
1625 ~~[25/password](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#password)~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#password>

1626 <SubjectConfirmationData>: *Base64 ( Password )*

1627 The subject of the assertion is the party that can present the password value specified in  
1628 <SubjectConfirmationData>.

1629 The username of the subject is specified by means of the <NameIdentifier> element.

1630 **7.1.6. Password (One-Way-Function SHA-1):**

1631 **URI:** ~~[http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/password-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/password-sha1)~~  
1632 ~~[sha1](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#password-sha1)~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#password-sha1>

1633 <SubjectConfirmationData>: *Base64 ( SHA1 ( Password ) )*

1634 The subject of the assertion is the party that can present the password such that the SHA1 digest of  
1635 the specified password matches the value specified in <SubjectConfirmationData>.

1636 The username of the subject is specified by means of the <NameIdentifier> element.

1637 **7.1.7. Kerberos**

1638 **URI:** urn:ietf:rfc:1510

1639 <SubjectConfirmationData>: A Kerberos Ticket

1640 The subject is authenticated by means of the Kerberos protocol [RFC 1510], an instantiation of the  
1641 Needham-Schroeder symmetric key authentication mechanism [Needham78].

1642 **7.1.8. SSL/TLS Certificate Based Client Authentication:**

1643 **URI:** urn:ietf:rfc:2246

1644 <ds:KeyInfo>: Any cryptographic key

1645 **7.1.9. Object Authenticator (SHA-1):**

1646 **URI:** ~~[http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/object-](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/object-sha1)~~  
1647 ~~[sha1](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#object-sha1)~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#object-sha1>

1648 <SubjectConfirmationData>: *Base64 ( SHA1 ( Object ) )*

1649 This authenticator element is the result of computing a digest, using the SHA -1 hash algorithm. It is  
1650 used when the subject can be represented as a binary string, for example when it is an XML  
1651 document or the disk image of executable code. Any preprocessing of the subject prior to  
1652 computation of the digest is out of scope. The name of the subject should be conveyed in an  
1653 accompanying NameIdentifier element.

1654 **7.1.10. PKCS#7**

1655 **URI:** urn:ietf:rfc:2315

1656 <SubjectConfirmationData>: *Base64 ( PKCS#7 ( Object ) )*

1657 This authenticator element is signed data in PKCS#7 format [PKCS#7]. The posited identity of the  
1658 signer must be conveyed in an accompanying NameIdentifier element. This subject type may be  
1659 included in the subject field of an authentication query, in which case the corresponding response  
1660 indicates whether the posited signer is, indeed, the signer. It may be included in an attribute query,  
1661 in which case, the requested attribute values for the subject authenticated by the signed data are  
1662 returned. It may be included in an authorization query, in which case, the access request  
1663 represented by the signed data shall be identified by the accompanying object element, and the  
1664 corresponding authorization decision assertion indicates whether the signer is authorized for the  
1665 access request represented by the object element.

### 1666 **7.1.11. Cryptographic Message Syntax**

1667 **URI:** urn:ietf:rfc:2630

1668 <SubjectConfirmationData>: *Base64* ( CMS ( Object ))

1669 This authenticator element is signed data in CMS format [CMS]. See also 7.1.10

### 1670 **7.1.12. XML Digital Signature**

1671 **URI:** ~~urn:ietf:rfc:2630~~[urn:ietf:rfc:3075](http://www.ietf.org/rfc/rfc3075)

1672 <SubjectConfirmationData>: *Base64* ( XML-SIG ( Object ))

1673 <ds:KeyInfo>: A cryptographic signing key

1674 This authenticator element is signed data in XML Signature format. See also 7.1.10

## 1675 **7.2. Action Namespace Identifiers**

1676 The following identifiers MAY be used in the `ActionNamespace` attribute (see Section 2.4.4.1) to  
1677 refer to common sets of actions to perform on resources.

### 1678 **7.2.1. Read/Write/Execute/Delete/Control:**

1679 **URI:** ~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/rwede>~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#rweadc>

1681 Defined actions:

1682     Read Write Execute Delete Control

1683 These actions are interpreted in the normal manner, i.e.

1684 Read

1685     The subject may read the resource

1686 Write

1687     The subject may modify the resource

1688 Execute

1689     The subject may execute the resource

1690 Delete

1691     The subject may delete the resource

1692 Control

1693     The subject may specify the access control policy for the resource

## 1694 **7.2.2. Read/Write/Execute/Delete/Control with Negation:**

1695 **URI:** ~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/rwcdc->~~  
1696 ~~[negationhttp://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#rwcdc-negation](http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#rwcdc-negation)~~

1697 Defined actions:

1698 `Read Write Execute Delete Control ~Read ~Write ~Execute ~Delete ~Control`

1699 The actions specified in section 7.2.1 are interpreted in the same manner described there. Actions  
1700 prefixed with a tilde ~ are negated permissions and are used to affirmatively specify that the stated  
1701 permission is denied. Thus a subject described as being authorized to perform the action ~Read is  
1702 affirmatively denied read permission.

1703 An application MUST NOT authorize both an action and its negated form.

## 1704 **7.2.3. Get/Head/Put/Post:**

1705 **URI:** ~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/ghpp>~~  
1706 ~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#ghpp>~~

1707 Defined actions:

1708 `GET HEAD PUT POST`

1709 These actions bind to the corresponding HTTP operations. For example a subject authorized to  
1710 perform the GET action on a resource is authorized to retrieve it.

1711 The GET and HEAD actions loosely correspond to the conventional read permission and the PUT  
1712 and POST actions to the write permission. The correspondence is not exact however since a HTTP  
1713 GET operation may cause data to be modified and a POST operation may cause modification to a  
1714 resource other than the one specified in the request. For this reason a separate Action URI  
1715 specifier is provided.

## 1716 **7.2.4. UNIX File Permissions:**

1717 **URI:** ~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/unix>~~  
1718 ~~<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#unix>~~

1719 The defined actions are the set of UNIX file access permissions expressed in the numeric (octal)  
1720 notation.

1721 The action string is a four digit numeric code:

1722 *extended user group world*

1723 Where the *extended* access permission has the value

1724 +2 if sgid is set

1725 +4 if suid is set

1726 The *user group* and *world* access permissions have the value

1727 +1 if execute permission is granted

1728 +2 if write permission is granted

1729 +4 if read permission is granted

1730 For example 0754 denotes the UNIX file access permission: user read, write and execute, group  
1731 read and execute and world read.

1732

## 8. SAML Schema Listings

1733

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

1734

### 8.1. Assertion Schema

1735

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

1736

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-Baker
(VeriSign Inc.) -->
<schema
  targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-
sstc-schema-assertion-26.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema" xmlns:saml="http://www.oasis-
open.org/committees/security/docs/draft-sstc-schema-assertion-26.xsd"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  elementFormDefault="unqualified">
  <import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="xmldsig-core-schema.xsd"/>
  <annotation>
    <documentation>draft-sstc-schema-assertion-26.xsd</documentation>
  </annotation>
  <simpleType name="IDType">
    <restriction base="string"/>
  </simpleType>
  <simpleType name="IDReferenceType">
    <restriction base="string"/>
  </simpleType>
  <simpleType name="DecisionType">
    <restriction base="string">
      <enumeration value="Permit"/>
      <enumeration value="Deny"/>
      <enumeration value="Indeterminate"/>
    </restriction>
  </simpleType>
  <element name="AssertionSpecifier" type="saml:AssertionSpecifierType"/>
  <complexType name="AssertionSpecifierType">
    <choice>
      <element ref="saml:AssertionIDReference"/>
      <element ref="saml:Assertion"/>
    </choice>
  </complexType>
  <del><element name="AssertionID" type="saml:IDType"/></del><del>name="AssertionIDReference"
type="saml:IDReferenceType"/></del>
  <element name="Assertion" type="saml:AssertionType"/>
  <complexType name="AssertionType">
    <sequence>
      <element ref="saml:Conditions" minOccurs="0"/>
      <element ref="saml:Advice" minOccurs="0"/>
      <choice minOccurs="0" maxOccurs="unbounded">
        <element ref="saml:Statement"/>
        <element ref="saml:SubjectStatement"/>
        <element ref="saml:AuthenticationStatement"/>
        <element ref="saml:AuthorizationDecisionStatement"/>
        <element ref="saml:AttributeStatement"/>
      </choice>
      <del><element ref="ds:Signature" minOccurs="0"
maxOccurs="unbounded"/></del><del>"ds:Signature" minOccurs="0"/></del>
    </sequence>
  </complexType>
</schema>
```

1787

```

1788     <attribute name="MajorVersion" type="integer" use="required" />
1789     <attribute name="MinorVersion" type="integer" use="required" />
1790     <attribute name="AssertionID" type="saml:IDType" use="required" />
1791     <attribute name="Issuer" type="string" use="required" />
1792     <attribute name="IssueInstant" type="dateTime" use="required" />
1793 </complexType>
1794 <element name="Conditions" type="saml:ConditionsType" />
1795 <complexType name="ConditionsType">
1796     <choice minOccurs="0" maxOccurs="unbounded">
1797         <element ref="saml:Condition" />
1798         <element ref="saml:AudienceRestrictionCondition" />
1799     </choice>
1800     <attribute name="NotBefore" type="dateTime" use="optional" />
1801     <attribute name="NotOnOrAfter" type="dateTime" use="optional" />
1802 </complexType>
1803 <element name="Condition" type="saml:ConditionAbstractType" />
1804 <complexType name="ConditionAbstractType" abstract="true" />
1805 <element name="AudienceRestrictionCondition"
1806     type="saml:AudienceRestrictionConditionType" />
1807 <complexType name="AudienceRestrictionConditionType">
1808     <complexContent>
1809         <extension base="saml:ConditionAbstractType">
1810             <sequence>
1811                 <element ref="saml:Audience" maxOccurs="unbounded" />
1812             </sequence>
1813         </extension>
1814     </complexContent>
1815 </complexType>
1816 <element name="Audience" type="anyURI" />
1817 <element name="TargetRestrictionCondition"
1818     type="saml:TargetRestrictionConditionType" />
1819 <complexType name="TargetRestrictionConditionType">
1820     <complexContent>
1821         <extension base="saml:ConditionAbstractType">
1822             <sequence>
1823                 <element ref="saml:Target"
1824                     minOccurs="1" maxOccurs="unbounded" />
1825             </sequence>
1826         </extension>
1827     </complexContent>
1828 </complexType>
1829 <element name="Target" type="anyURI" />
1830 <element name="Advice" type="saml:AdviceType" />
1831 <complexType name="AdviceType">
1832     <sequence>
1833         <choice minOccurs="0" maxOccurs="unbounded">
1834             <element ref="saml:AssertionSpecifier" />
1835             <element ref="saml:AdviceElement" />
1836             <any namespace="##other" processContents="lax" />
1837         </choice>
1838     </sequence>
1839 </complexType>
1840 <element name="AdviceElement" type="saml:AdviceAbstractType" />
1841 <complexType name="AdviceAbstractType">
1842     <element name="Statement" type="saml:StatementAbstractType" />
1843 <complexType name="StatementAbstractType" abstract="true" />
1844 <element name="SubjectStatement" type="saml:SubjectStatementAbstractType" />
1845 <complexType name="SubjectStatementAbstractType" abstract="true">
1846     <complexContent>
1847         <extension base="saml:StatementAbstractType">
1848             <sequence>
1849                 <element ref="saml:Subject" />
1850             </sequence>

```

```

1851     </extension>
1852   </complexContent>
1853 </complexType>
1854 <element name="Subject" type="saml:SubjectType" />
1855 <complexType name="SubjectType">
1856   <choice maxOccurs="unbounded">
1857     <sequence>
1858       <element ref="saml:NameIdentifier" />
1859       <element ref="saml:SubjectConfirmation" minOccurs="0" />
1860     </sequence>
1861     <element ref="saml:SubjectConfirmation" />
1862   </choice>
1863 </complexType>
1864 <element name="NameIdentifier" type="saml:NameIdentifierType" />
1865 <complexType name="NameIdentifierType">
1866   <attribute name="SecurityDomain" type="string" />
1867   <attribute name="Name" type="string" />
1868 </complexType>
1869 <element name="SubjectConfirmation" type="saml:SubjectConfirmationType" />
1870 <complexType name="SubjectConfirmationType">
1871   <sequence>
1872     <element ref="saml:ConfirmationMethod" maxOccurs="unbounded" />
1873     <element ref="saml:SubjectConfirmationData" minOccurs="0" />
1874     <element ref="ds:KeyInfo" minOccurs="0" />
1875   </sequence>
1876 </complexType>
1877 <element name="SubjectConfirmationData" type="string" minOccurs="0" />
1878 <element name="ConfirmationMethod" type="anyURI" />
1879 <element name="AuthenticationStatement"
1880   type="saml:AuthenticationStatementType" />
1881 <complexType name="AuthenticationStatementType">
1882   <complexContent>
1883     <extension base="saml:SubjectStatementAbstractType">
1884       <sequence>
1885         <element ref="saml:AuthenticationLocality" minOccurs="0" />
1886         <element ref="saml:AuthorityBinding"
1887           minOccurs="0" maxOccurs="unbounded" />
1888       </sequence>
1889       <attribute name="AuthenticationMethod" type="anyURI" />
1890       <attribute name="AuthenticationInstant" type="dateTime" />
1891     </extension>
1892   </complexContent>
1893 </complexType>
1894 <element name="AuthenticationLocality"
1895   type="saml:AuthenticationLocalityType" />
1896 <complexType name="AuthenticationLocalityType">
1897   <attribute name="IPAddress" type="string" use="optional" />
1898   <attribute name="DNSAddress" type="string" use="optional" />
1899 </complexType>
1900 <element name="AuthorityBinding" type="saml:AuthorityBindingType" />
1901 <complexType name="AuthorityBindingType">
1902   <attribute name="AuthorityKind" type="saml:AuthorityKindType" />
1903   <attribute name="Location" type="anyURI" use="required" />
1904   <attribute name="Binding" type="anyURI" use="required" />
1905 </complexType>
1906 <simpleType name="AuthorityKindType">
1907   <restriction base="string">
1908     <enumeration value="authentication" />
1909     <enumeration value="attribute" />
1910     <enumeration value="authorization" />
1911   </restriction>
1912 </simpleType>
1913 <element name="AuthorizationDecisionStatement"

```

```

1914         type="saml:AuthorizationDecisionStatementType" />
1915     <complexType name="AuthorizationDecisionStatementType">
1916         <complexContent>
1917             <extension base="saml:SubjectStatementAbstractType">
1918                 <sequence>
1919                     <element ref="saml:Actions" />
1920                     <element ref="saml:Evidence"
1921                         minOccurs="0" maxOccurs="unbounded" />
1922                 </sequence>
1923                 <attribute name="Resource" type="anyURI" use="optional" />
1924                 <attribute name="Decision"
1925                     type="saml:DecisionType" use="optional" />
1926             </extension>
1927         </complexContent>
1928     </complexType>
1929     <element name="Actions" type="saml:ActionsType" />
1930     <complexType name="ActionsType">
1931         <sequence>
1932             <element ref="saml:Action" maxOccurs="unbounded" />
1933         </sequence>
1934         <attribute name="Namespace" type="anyURI" use="optional" />
1935     </complexType>
1936     <element name="Action" type="string" />
1937     <element name="Evidence" type="saml:AssertionSpecifierType" />
1938     <element name="AttributeStatement" type="saml:AttributeStatementType" />
1939     <complexType name="AttributeStatementType">
1940         <complexContent>
1941             <extension base="saml:SubjectStatementAbstractType">
1942                 <sequence>
1943                     <element ref="saml:Attribute" maxOccurs="unbounded" />
1944                 </sequence>
1945             </extension>
1946         </complexContent>
1947     </complexType>
1948     <element name="AttributeDesignator" type="saml:AttributeDesignatorType" />
1949     <complexType name="AttributeDesignatorType">
1950         <attribute name="AttributeName" type="string" />
1951         <attribute name="AttributeNamespace" type="anyURI" />
1952     </complexType>
1953     <element name="Attribute" type="saml:AttributeType" />
1954     <complexType name="AttributeType">
1955         <complexContent>
1956             <extension base="saml:AttributeDesignatorType">
1957                 <sequence>
1958                     <element ref="saml:AttributeValue" maxOccurs="unbounded" />
1959                 </sequence>
1960             </extension>
1961         </complexContent>
1962     </complexType>
1963     <element name="AttributeValue"
1964         type="saml:AttributeValueType" /> type="saml:anyType" />
1965     <complexType name="AttributeValueType">
1966     <sequence>
1967     <any namespace="##any" processContents="lax"
1968     minOccurs="0" maxOccurs="unbounded" />
1969     </sequence>
1970     </complexType>
1971 </schema>
1972

```

1973

## 8.2. Protocol Schema

1974

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

1975

```
<?xml version="1.0" encoding="UTF-8"?>
```

1976

```
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-Baker  
(VeriSign Inc.) -->
```

1977

```
<schema
```

1978

```
  targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-
```

1979

```
sstc-schema-protocol-26.xsd"
```

1980

```
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-
```

1981

```
schema-assertion-26.xsd"
```

1982

```
  xmlns:samlp="http://www.oasis-open.org/committees/security/docs/draft-sstc-
```

1983

```
schema-protocol-26.xsd"
```

1984

```
  xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
```

1985

```
  <import
```

1986

```
    namespace="http://www.oasis-open.org/committees/security/docs/draft-sstc-
```

1987

```
schema-assertion-26.xsd"
```

1988

```
    schemaLocation="draft-sstc-schema-assertion-26.xsd" />
```

1989

```
  <import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="xmldsig-core-schema.xsd" />
```

1990

```
  <annotation>
```

1991

```
    <documentation>draft-sstc-schema-protocol-26.xsd</documentation>
```

1992

```
  </annotation>
```

1993

```
  <complexType name="RequestAbstractType" abstract="true">
```

1994

```
    <sequence>
```

1995

```
      <element ref="samlp:RespondWith"
```

1996

```
        minOccurs="0" maxOccurs="unbounded" />
```

1997

```
      <element ref="ds:Signature" minOccurs="0"
maxOccurs="unbounded" />
```

1998

```
      <sequence>
```

1999

```
        <attribute name="RequestID" type="saml:IDType" use="required" />
```

2000

```
        <attribute name="MajorVersion" type="integer" use="required" />
```

2001

```
        <attribute name="MinorVersion" type="integer" use="required" />
```

2002

```
        <attribute name="IssueInstant" type="dateTime" use="required" />
```

2003

```
      </sequence>
```

2004

```
    <element name="RespondWith" type="anyURI" />
```

2005

```
    <element name="Request" type="samlp:RequestType" />
```

2006

```
    <complexType name="RequestType">
```

2007

```
      <complexContent>
```

2008

```
        <extension base="samlp:RequestAbstractType">
```

2009

```
          <choice>
```

2010

```
            <element ref="samlp:Query" />
```

2011

```
            <element ref="samlp:SubjectQuery" />
```

2012

```
            <element ref="samlp:AuthenticationQuery" />
```

2013

```
            <element ref="samlp:AttributeQuery" />
```

2014

```
            <element ref="samlp:AuthorizationDecisionQuery" />
```

2015

```
            <element ref="saml:AssertionID" maxOccurs="unbounded" />
```

2016

```
            <element ref="samlp:AssertionArtifact" maxOccurs="unbounded" />
```

2017

```
          </choice>
```

2018

```
        </extension>
```

2019

```
      </complexContent>
```

2020

```
    </complexType>
```

2021

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

2022

```
  <element name="Query" type="samlp:QueryAbstractType" />
```

2023

```
  <complexType name="QueryAbstractType" abstract="true" />
```

2024

```
  <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType" />
```

2025

```
  <complexType name="SubjectQueryAbstractType" abstract="true">
```

2026

```
    <complexContent>
```

2027

```
      <extension base="samlp:QueryAbstractType">
```

2028

```
        <sequence>
```

2029

```
          <sequence>
```

2030

```
            <sequence>
```

2031

```
              <sequence>
```

2032

```
                <sequence>
```

```

2033         <element ref="saml:Subject" />
2034     </sequence>
2035 </extension>
2036 </complexContent>
2037 </complexType>
2038 <element name="AuthenticationQuery" type="samlp:AuthenticationQueryType" />
2039 <complexType name="AuthenticationQueryType">
2040 <complexContent>
2041     <extension base="samlp:SubjectQueryAbstractType">
2042     <sequence>
2043         <element ref="saml:ConfirmationMethod" minOccurs="0" />
2044     </sequence>
2045 </extension>
2046 </complexContent>
2047 </complexType>
2048 <element name="AttributeQuery" type="samlp:AttributeQueryType" />
2049 <complexType name="AttributeQueryType">
2050 <complexContent>
2051     <extension base="samlp:SubjectQueryAbstractType">
2052     <sequence>
2053         <element ref="saml:AttributeDesignator"
2054             minOccurs="0" maxOccurs="unbounded" />
2055     </sequence>
2056 </extension>
2057 </complexContent>
2058 </complexType>
2059 <element name="AuthorizationDecisionQuery"
2060     type="samlp:AuthorizationDecisionQueryType" />
2061 <complexType name="AuthorizationDecisionQueryType">
2062 <complexContent>
2063     <extension base="samlp:SubjectQueryAbstractType">
2064     <sequence>
2065         <element ref="saml:Actions" />
2066         <element ref="saml:Evidence"
2067             minOccurs="0" maxOccurs="unbounded" />
2068     </sequence>
2069     <attribute name="Resource" type="anyURI" use="required" />
2070 </extension>
2071 </complexContent>
2072 </complexType>
2073 <complexType name="ResponseAbstractType" abstract="true">
2074 <sequence>
2075     <element ref = "ds:Signature" minOccurs="0"
2076     maxOccurs="unbounded" />"ds:Signature" minOccurs="0" />
2077 </sequence>
2078     <attribute name="ResponseID" type="saml:IDType" use="required" />
2079     <attribute name="InResponseTo" type="saml:IDType" use="required" />
2080 type="saml:IDReferenceType"
2081 use="required" />
2082     <attribute name="MajorVersion" type="integer" use="required" />
2083     <attribute name="MinorVersion" type="integer" use="required" />
2084     <attribute name="IssueInstant" type="dateTime" use="required" />
2085 </complexContent>
2086 </complexType>
2087 <element name="Response" type="samlp:ResponseType" />
2088 <complexType name="ResponseType">
2089 <complexContent>
2090     <extension base="samlp:ResponseAbstractType">
2091     <sequence>
2092         <element ref="samlp:Status" />
2093         <element ref="saml:Assertion"
2094             minOccurs="0" maxOccurs="unbounded" />
2095     </sequence>

```

```

2096     </extension>
2097   </complexContent>
2098 </complexType>
2099 <element name="Status" type="samlp:StatusType" />
2100 <complexType name="StatusType">
2101   <sequence>
2102     <element ref="samlp:StatusCode" />
2103     <element ref="samlp:StatusMessage"
2104       minOccurs="0" maxOccurs="unbounded" />
2105     <element ref="samlp:StatusDetail" minOccurs="0" />
2106   </sequence>
2107 </complexType>
2108 <element name="StatusCode" type="samlp:StatusCodeType" />
2109 <complexType name="StatusCodeType">
2110   <sequence>
2111     <element ref="samlp:SubStatusCode" minOccurs="0" />
2112   </sequence>
2113   <attribute name="Value" type="samlp:StatusCodeEnumType" use="required" />
2114 </complexType>
2115 <simpleType name="StatusCodeEnumType">
2116   <restriction base="QName">
2117     <enumeration value="samlp:Success" />
2118     <enumeration value="samlp:VersionMismatch" />
2119     <enumeration value="samlp:Receiver" />
2120     <enumeration value="samlp:Sender" />
2121   </restriction>
2122 </simpleType>
2123 <element name="SubStatusCode" type="samlp:SubStatusCodeType" />
2124 <complexType name="SubStatusCodeType">
2125   <sequence>
2126     <element ref="samlp:SubStatusCode" minOccurs="0" />
2127   </sequence>
2128   <attribute name="Value" type="QName" use="required" />
2129 </complexType>
2130 <element name="StatusMessage" type="string" />
2131 <element name="StatusDetail" type="samlp:StatusDetailType" />
2132 <complexType name="StatusDetailType">
2133   <sequence>
2134     <any namespace="##any"
2135       processContents="lax" minOccurs="0" maxOccurs="unbounded" />
2136   </sequence>
2137 </complexType>
2138 </schema>
2139

```

## 9. References

2140

- 2141 [Needham78] R. Needham et al., *Using Encryption for Authentication in Large Networks*  
2142 *of Computers*, Communications of the ACM, Vol. 21 (12), pp. 993-999,  
2143 December 1978.
- 2144 [Kern-84] B. Kernighan, Rob Pike *The UNIX Programming Environment*, (March  
2145 1984) Prentice Hall Computer Books;
- 2146 [PKCS1] B. Kaliski, *PKCS #1: RSA Encryption Version 2.0*, RSA Laboratories, also  
2147 IETF RFC 2437, October 1998. <http://www.ietf.org/rfc/rfc2437.txt>
- 2148 [PKCS7] B. Kaliski., "PKCS #7: Cryptographic Message Syntax, Version 1.5.", RFC  
2149 2315, March 1998.
- 2150 [RFC 1510] J. Kohl, C. Neuman. *The Kerberos Network Authentication*  
2151 *Service Requestor* (V5). September 1993. <http://www.ietf.org/rfc/rfc1510.txt>
- 2152 [RFC 2246] T. Dierks, C. Allen. *The TLS Protocol Version 1.0*. January 1999.  
2153 <http://www.ietf.org/rfc/rfc2246.txt>
- 2154 [RFC 2630] R. Housley. *Cryptographic Message Syntax*. June 1999.  
2155 <http://www.ietf.org/rfc/rfc630.txt>
- 2156 [RFC 2648] R. Moats. *A URN Namespace for IETF Documents*. August 1999.  
2157 <http://www.ietf.org/rfc/rfc2648.txt>
- 2158 [RFC 3075] D. Eastlake, J. Reagle, D. Solo. *XML-Signature Syntax and Processing*.  
2159 March 2001. <http://www.ietf.org/rfc/rfc3075.txt>
- 2160 [RFC2104] H. Krawczyk et al., *HMAC: Keyed Hashing for Message Authentication*,  
2161 <http://www.ietf.org/rfc/rfc2104.txt>, IETF RFC 2104, February 1997.
- 2162 [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,  
2163 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997
- 2164 [SAMLBind] P. Mishra et al., *Bindings and Profiles for the OASIS Security Assertion*  
2165 *Markup Language (SAML)*, [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-bindings-model-07.pdf)  
2166 [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),  
2167 OASIS, December 2001.
- 2168 [SAMLConform] **TBS**
- 2169 [SAMLGloss] J. Hodges et al., *Glossary for the OASIS Security Assertion Markup*  
2170 *Language (SAML)*, [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-glossary-02.pdf)  
2171 [open.org/committees/security/docs/draft-sstc-glossary-02.pdf](http://www.oasis-open.org/committees/security/docs/draft-sstc-glossary-02.pdf), OASIS,  
2172 December 2001.
- 2173 [SAML-XSD] 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)  
2174 [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),  
2175 OASIS, December 2001.
- 2176 [SAMLSecure] **TBS**
- 2177 [SAML-XSD] P. Hallam-Baker et al., *SAML assertion schema*, [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-21.xsd)  
2178 [open.org/committees/security/docs/draft-sstc-schema-assertion-21.xsd](http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-21.xsd),  
2179 OASIS, December 2001.
- 2180 [Schema1] H. S. Thompson et al., *XML Schema Part 1: Structures*,  
2181 <http://www.w3.org/TR/xmlschema-1/>, World Wide Web Consortium  
2182 Recommendation, May 2001.
- 2183 [Schema2] P. V. Biron et al., *XML Schema Part 2: Datatypes*,  
2184 <http://www.w3.org/TR/xmlschema-2/>, World Wide Web Consortium  
2185 Recommendation, May 2001.
- 2186 [XMLEnc] *XML Encryption Specification*, In development.

- 2187 **[XMLSig]** D. Eastlake et al., *XML-Signature Syntax and Processing*,  
2188 <http://www.w3.org/TR/xmlsig-core/>, World Wide Web Consortium.
- 2189 **[XMLSig-XSD]** XML Signature Schema available from <http://www.w3.org/TR/2000/CR-xmlsig-core-20001031/xmlsig-core-schema.xsd>  
2190
- 2191 **[XTAML]** P. Hallam-Baker, *XML Trust Axiom Markup Language 1.0*,  
2192 <http://www.xmltrustcenter.org/>, VeriSign Inc. September 2001.
- 2193 **[W3C-CHAR]** <http://www.w3.org/TR/WD-charreq>
- 2194 **[UNICODE-C]** <http://www.unicode.org/unicode/reports/tr15/tr15-21.html>
- 2195 **[W3C-CharMod]** <http://www.w3.org/TR/charmod/>
- 2196 **[XML]** <http://www.w3.org/TR/REC-xml>

2197

## Appendix A. Notices

2198 OASIS takes no position regarding the validity or scope of any intellectual property or other rights  
2199 that might be claimed to pertain to the implementation or use of the technology described in this  
2200 document or the extent to which any license under such rights might or might not be available;  
2201 neither does it represent that it has made any effort to identify any such rights. Information on  
2202 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS  
2203 website. Copies of claims of rights made available for publication and any assurances of licenses to  
2204 be made available, or the result of an attempt made to obtain a general license or permission for  
2205 the use of such proprietary rights by implementors or users of this specification, can be obtained  
2206 from the OASIS Executive Director.

2207 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
2208 applications, or other proprietary rights which may cover technology that may be required to  
2209 implement this specification. Please address the information to the OASIS Executive Director.

2210 Copyright © The Organization for the Advancement of Structured Information Standards [OASIS]  
2211 2001. All Rights Reserved.

2212 This document and translations of it may be copied and furnished to others, and derivative works  
2213 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
2214 published and distributed, in whole or in part, without restriction of any kind, provided that the above  
2215 copyright notice and this paragraph are included on all such copies and derivative works. However,  
2216 this document itself may not be modified in any way, such as by removing the copyright notice or  
2217 references to OASIS, except as needed for the purpose of developing OASIS specifications, in  
2218 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights  
2219 document must be followed, or as required to translate it into languages other than English.

2220 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
2221 successors or assigns.

2222 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
2223 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
2224 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY  
2225 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
2226 PARTICULAR PURPOSE.