



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  
36

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

**Document identifier:** draft-sstc-core-25

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

**Publication date:** January 10th 2002

**Maturity Level:** Committee Working Draft

**Send comments to:** [security-services-comment@lists.oasis-open.org](mailto:security-services-comment@lists.oasis-open.org) *unless* you are subscribed to the security-services list for committee members -- send comments there if so.  
Note: Before sending messages to the security-services-comment list, you must first subscribe. To subscribe, send an email message to [security-services-comment-request@lists.oasis-open.org](mailto:security-services-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  
Irving Reid, Baltimore  
Krishna Sankar, Cisco Systems Inc

36		
37	<b>ASSERTIONS AND PROTOCOL FOR THE OASIS SECURITY ASSERTION MARKUP</b>	
38	<b>LANGUAGE (SAML)</b>	<b>1</b>
39	<b>1. INTRODUCTION</b>	<b>6</b>
40	1.1. NOTATION	6
41	1.2. SCHEMA ORGANIZATION AND NAMESPACES	6
42	1.3. SAML CONCEPTS (NON-NORMATIVE)	7
43	1.3.1. <i>Overview</i>	7
44	1.3.2. <i>SAML and URI-Based Identifiers</i>	8
45	1.3.3. <i>SAML and Extensibility</i>	9
46	<b>2. SAML ASSERTIONS</b>	<b>10</b>
47	2.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS	10
48	2.2. SIMPLE TYPES	10
49	2.2.1. <i>Simple Type IDType</i>	10
50	2.2.2. <i>Simple Type DecisionType</i>	11
51	2.3. ASSERTIONS	11
52	2.3.1. <i>Element &lt;AssertionSpecifier&gt;</i>	11
53	2.3.2. <i>Element &lt;AssertionID&gt;</i>	11
54	2.3.3. <i>Element &lt;Assertion&gt;</i>	12
55	2.3.3.1. <i>Element &lt;Conditions&gt;</i>	13
56	2.3.3.1.1 <i>Attributes NotBefore and NotOnOrAfter</i>	14
57	2.3.3.1.2 <i>Element &lt;Condition&gt;</i>	14
58	2.3.3.1.3 <i>Elements &lt;AudienceRestrictionCondition&gt; and &lt;Audience&gt;</i>	14
59	2.3.3.1.4 <i>Condition Type TargetRestrictionType</i>	15
60	2.3.3.2. <i>Elements &lt;Advice&gt; and &lt;AdviceElement&gt;</i>	15
61	2.4. STATEMENTS	16
62	2.4.1. <i>Element &lt;Statement&gt;</i>	16
63	2.4.2. <i>Element &lt;SubjectStatement&gt;</i>	16

64	2.4.2.1. Element <Subject>	16
65	2.4.2.2. Element <NameIdentifier>	17
66	2.4.2.3. Elements <SubjectConfirmation>, <ConfirmationMethod>, and <SubjectConfirmationData>	17
67	<b>2.4.3. Element &lt;AuthenticationStatement&gt;</b>	<b>18</b>
68	2.4.3.1. Element <AuthenticationLocality>	18
69	2.4.3.2. Element <AuthorityBinding>	19
70	<b>2.4.4. Element &lt;AuthorizationDecisionStatement&gt;</b>	<b>19</b>
71	2.4.4.1. Elements <Actions> and <Action>	20
72	2.4.4.2. Element <Evidence>	20
73	<b>2.4.5. Element &lt;AttributeStatement&gt;</b>	<b>20</b>
74	2.4.5.1. Elements <AttributeDesignator> and <Attribute>	21
75	2.4.5.1.1 Element <AttributeValue>	21
76	<b>3. SAML PROTOCOL</b>	<b>23</b>
77	<b>3.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS</b>	<b>23</b>
78	<b>3.2. REQUESTS</b>	<b>23</b>
79	<b>3.2.1. Complex Type RequestAbstractType</b>	<b>23</b>
80	3.2.1.1. Element <RespondWith>	24
81	<b>3.2.2. Element &lt;Request&gt;</b>	<b>24</b>
82	<b>3.3. QUERIES</b>	<b>25</b>
83	<b>3.3.1. Element &lt;Query&gt;</b>	<b>25</b>
84	<b>3.3.2. Element &lt;SubjectQuery&gt;</b>	<b>25</b>
85	<b>3.3.3. Element &lt;AuthenticationQuery&gt;</b>	<b>26</b>
86	<b>3.3.4. Element &lt;AttributeQuery&gt;</b>	<b>26</b>
87	<b>3.3.5. Element &lt;AuthorizationDecisionQuery&gt;</b>	<b>27</b>
88	<b>3.4. RESPONSES</b>	<b>27</b>
89	<b>3.4.1. Complex Type ResponseAbstractType</b>	<b>27</b>
90	<b>3.4.2. Element &lt;Response&gt;</b>	<b>28</b>
91	<b>3.4.3. Element &lt;Status&gt;</b>	<b>28</b>
92	3.4.3.1. Element <StatusCode>	29
93	3.4.3.2. Element <StatusMessage>	29
94	3.4.3.3. Element <StatusDetail>	30

95	<b>3.4.4. Simple Type StatusCodeType</b>	31
96	<b>4. SAML VERSIONING</b>	<b>32</b>
97	4.1. ASSERTION VERSION	32
98	4.2. REQUEST VERSION	32
99	4.3. RESPONSE VERSION	33
100	<b>5. SAML &amp; XML-SIGNATURE SYNTAX AND PROCESSING</b>	<b>34</b>
101	5.1. SIGNING ASSERTIONS	34
102	5.2. REQUEST/RESPONSE SIGNING	34
103	5.3. SIGNATURE INHERITANCE (A.K.A. SUPER-SIGNATURES & SUB-MESSAGES)	35
104	5.3.1. <i>Rationale</i>	35
105	5.3.2. <i>Rules for SAML Signature Inheritance</i>	35
106	5.4. XML SIGNATURE PROFILE	35
107	5.4.1. <i>Signing formats</i>	35
108	5.4.2. <i>CanonicalizationMethod</i>	35
109	5.4.3. <i>Transforms</i>	35
110	5.4.4. <i>KeyInfo</i>	36
111	5.4.5. <i>Binding between statements in a multi-statement assertion</i>	36
112	5.4.6. <i>Security considerations</i>	36
113	5.4.6.1. <i>Replay Attack</i>	36
114	<b>6. SAML EXTENSIONS</b>	<b>37</b>
115	6.1. ASSERTION SCHEMA EXTENSION	37
116	6.2. PROTOCOL SCHEMA EXTENSION	37
117	6.3. USE OF TYPE DERIVATION AND SUBSTITUTION GROUPS	38
118	<b>7. SAML-DEFINED IDENTIFIERS</b>	<b>39</b>

119	<b>7.1. CONFIRMATION METHOD IDENTIFIERS</b>	39
120	<b>7.1.1. SAML Artifact:</b>	39
121	<b>7.1.2. SAML Artifact (SHA-1):</b>	39
122	<b>7.1.3. Holder of Key:</b>	39
123	<b>7.1.4. Sender Vouches:</b>	39
124	<b>7.1.5. Password (Pass-Through):</b>	39
125	<b>7.1.6. Password (One-Way-Function SHA-1):</b>	40
126	<b>7.1.7. Kerberos</b>	40
127	<b>7.1.8. SSL/TLS Certificate Based Client Authentication:</b>	40
128	<b>7.1.9. Object Authenticator (SHA-1):</b>	40
129	<b>7.1.10. PKCS#7</b>	40
130	<b>7.1.11. Cryptographic Message Syntax</b>	41
131	<b>7.1.12. XML Digital Signature</b>	41
132	<b>7.2. ACTION NAMESPACE IDENTIFIERS</b>	41
133	<b>7.2.1. Read/Write/Execute/Delete/Control:</b>	41
134	<b>7.2.2. Read/Write/Execute/Delete/Control with Negation:</b>	41
135	<b>7.2.3. Get/Head/Put/Post:</b>	42
136	<b>7.2.4. UNIX File Permissions:</b>	42
137	<b>8. SAML SCHEMA LISTINGS</b>	43
138	<b>8.1. ASSERTION SCHEMA</b>	43
139	<b>8.2. PROTOCOL SCHEMA</b>	46
140	<b>9. REFERENCES</b>	50
141	<b>APPENDIX A. NOTICES</b>	52
142		

# 143 1. Introduction

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

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

## 151 1.1. Notation

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

155 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",  
156 "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be  
157 interpreted as described in IETF RFC 2119 **[RFC2119]**:

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

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

164 Listings of SAML schemas appear like this.

165 `Example code listings appear like this.`

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

- 170 • The prefix `saml`: stands for the SAML assertion namespace.
- 171 • The prefix `samlp`: stands for the SAML request-response protocol namespace.
- 172 • The prefix `ds`: stands for the W3C XML Signature namespace.
- 173 • The prefix `xsd`: stands for the W3C XML Schema namespace in example listings. In  
174 schema listings, this is the default namespace and no prefix is shown.

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

## 177 1.2. Schema Organization and Namespaces

178 The SAML assertion structures are defined in a schema **[SAML-XSD]** associated with the following  
179 XML namespace:`[PHB1]`

180 `http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-25.xsd`

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

183 `http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-25.xsd`



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

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

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

189 The XML Signature element `<ds:KeyInfo>`, defined in [**XMLSig**] §4.4, is of particular interest in  
190 SAML.

## 191 **1.3. SAML Concepts (Non-Normative)**

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

### 195 **1.3.1. Overview**

196 The Security Assertion Markup Language (SAML) is an XML-based framework for exchanging  
197 security information. This security information is expressed in the form of assertions about subjects,  
198 where a subject is an entity (either human or computer) that has an identity in some security  
199 domain. A typical example of a subject is a person, identified by his or her email address in a  
200 particular Internet domain[PHB2].

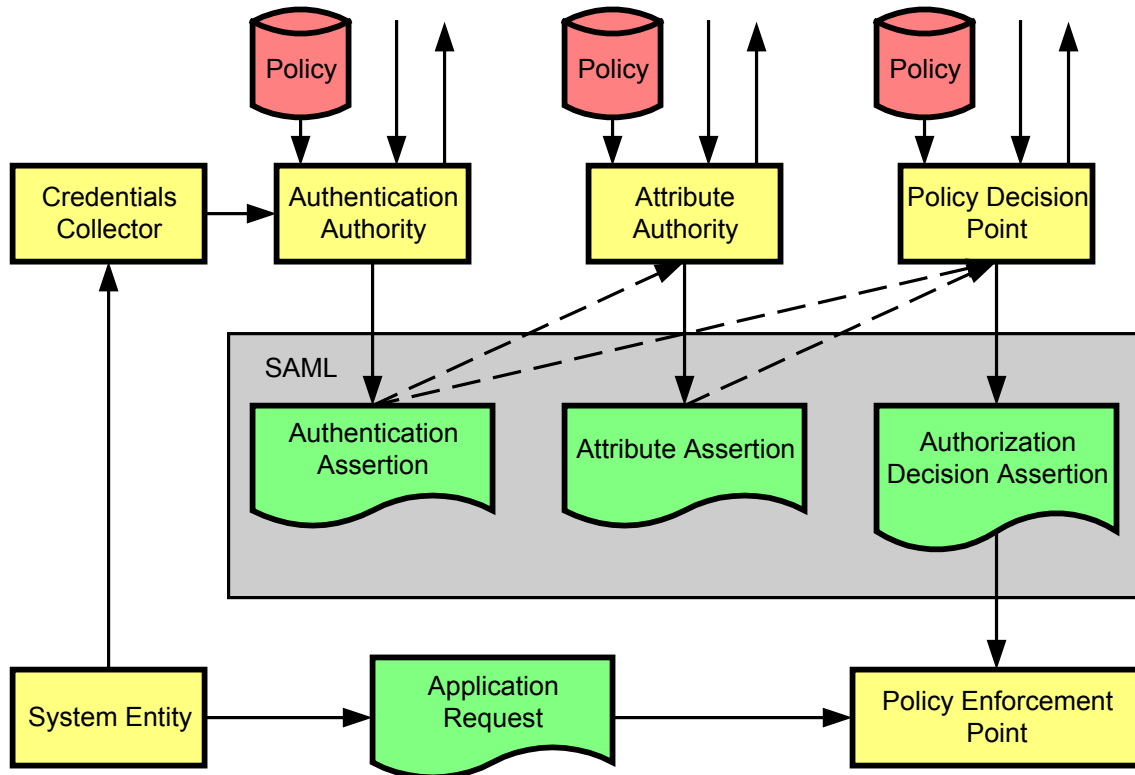
201 Assertions can convey information about authentication acts performed by subjects, attributes of  
202 subjects, and authorization decisions about whether subjects are allowed to access certain  
203 resources. Assertions are represented as XML constructs and have a nested structure, whereby a  
204 single assertion might contain several different internal statements about authentication,  
205 authorization, and attributes. Note that authentication assertions merely describe acts of  
206 authentication that happened previously; checking and revoking of credentials is outside the scope  
207 of this version of SAML[PHB3].

208 Assertions are issued by SAML authorities, namely, authentication authorities, attribute authorities,  
209 and policy decision points. SAML provides a protocol by which clients can request assertions from  
210 SAML authorities and get a response from them. This protocol, consisting of XML-based request  
211 and response message formats, can be bound to many different underlying communications and  
212 transport protocols; SAML currently defines one binding, to SOAP over HTTP. It is possible to  
213 produce and consume SAML assertions without using the SAML protocol, although interoperability  
214 is likely to be harmed in this case[PHB4].

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

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





220

221

**Figure 1 The SAML Domain Model**

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

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

### 230 **1.3.2. SAML and URI-Based Identifiers**

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

234 **urn:ietf:rfc:1510**

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

237 **http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/rwcdc**

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



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

248 See 7 for a list of SAML-defined identifiers.

### 249 **1.3.3. SAML and Extensibility**

250 The XML formats for SAML assertions and protocol messages have been designed to support  
251 extension[PHB5].

252 However, it is possible that the use of extensions will harm interoperability and the use of  
253 extensions SHOULD be carefully considered.



## 254 2. SAML Assertions

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

- 257 • **Authentication:** The specified subject was authenticated by a particular means at a  
258 particular time.
- 259 • **Authorization Decision:** A request to allow the specified subject to access the specified  
260 resource has been granted or denied.
- 261 • **Attribute:** The specified subject is associated with the supplied attributes.

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

### 265 2.1. Schema Header and Namespace Declarations

266 The following schema fragment defines the XML namespaces and other header information for the  
267 assertion schema:[PHB6]

```
268 <schema  
269   targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-  
270 sstc-schema-assertion-25.xsd"  
271   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
272   xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-  
273 schema-assertion-25.xsd"  
274   xmlns="http://www.w3.org/2001/XMLSchema"  
275   elementFormDefault="unqualified">  
276   <import namespace="http://www.w3.org/2000/09/xmldsig#"  
277     schemaLocation="xmldsig-core-schema.xsd"/>  
278   <annotation  
279     <documentation>draft-sstc-schema-assertion-25.xsd</documentation>  
280   </annotation>  
281   ...  
282 </schema>
```

### 283 2.2. Simple Types

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

#### 285 2.2.1. Simple Type IDType

286 The **IDType** simple type is used to declare and reference identifiers to assertions, requests, and  
287 responses.

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

- 289 • Any party that assigns an identifier MUST ensure that there is negligible probability that that  
290 party or any other party will assign the same identifier to a different data object.
- 291 • Where a data object declares that it has a particular identifier, there MUST be exactly one  
292 such declaration.

293 The mechanism by which the application ensures that the identifier is unique is left to the  
294 implementation. In the case that a pseudorandom technique is employed, the probability of two  
295 randomly chosen identifiers being identical MUST be less than  $2^{-128}$  and SHOULD be less than  
296  $2^{-160}$ .

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

300 The following schema fragment defines the **IDType** simple type:

```
301 <simpleType name="IDType">  
302   <restriction base="string"/>  
303 </simpleType>
```

## 304 2.2.2. Simple Type DecisionType

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

307 Permit

308       The specified action is permitted.

309 Deny

310       The specified action is denied.

311 Indeterminate

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

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

```
314 <simpleType name="DecisionType">  
315   <restriction base="string">  
316     <enumeration value="Permit"/>  
317     <enumeration value="Deny"/>  
318     <enumeration value="Indeterminate"/>  
319   </restriction>  
320 </simpleType>
```

## 321 2.3. Assertions

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

### 323 2.3.1. Element <AssertionSpecifier>

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

326 <AssertionID>

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

328 <Assertion>

329       Specifies an assertion by value.

330 The following schema fragment defines the <AssertionSpecifier> element and its

331 **AssertionSpecifierType** complex type:

```
332 <element name="AssertionSpecifier" type="saml:AssertionSpecifierType"/>  
333 <complexType name="AssertionSpecifierType">  
334   <choice>  
335     <element ref="saml:AssertionID"/>  
336     <element ref="saml:Assertion"/>  
337   </choice>  
338 </complexType>
```

### 339 2.3.2. Element <AssertionID>

340 The <AssertionID> element makes a reference to a SAML assertion by means of the value the  
341 assertion's `AssertionID` attribute.

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

```
343 <element name="AssertionID" type="saml:IDType"/>
```

### 344 2.3.3. Element <Assertion>

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

348 MajorVersion [Required]

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

351 MinorVersion [Required]

352 The minor version of this assertion. The identifier for the version of SAML defined in this  
353 specification is 0. Processing of this attribute is specified in Section 3.4.2.

354 AssertionID [Required]

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

357 Issuer [Required]

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

361 IssueInstant [Required]

362 The time instant of issue. It has the type **dateTime**, which is built in to the W3C XML  
363 Schema Datatypes specification [**Schema2**].

364 <Conditions> [Optional]

365 Conditions that MUST be taken into account in assessing the validity of the assertion.

366 <Advice> [Optional]

367 Additional information related to the assertion that assists processing in certain situations  
368 but which MAY be ignored by applications that do not support its use.

369 One or more of the following statement elements:

370 <Statement>

371 A statement defined in an extension schema.

372 <SubjectStatement>

373 A subject statement defined in an extension schema.

374 <AuthenticationStatement>

375 An authentication statement.

376 <AuthorizationDecisionStatement>

377 An authorization decision statement.

378 <AttributeStatement>

379 An attribute statement.

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

```
382 <complexType name="AssertionType">  
383   <sequence>  
384     <element ref="saml:Conditions" minOccurs="0"/>  
385     <element ref="saml:Advice" minOccurs="0"/>  
386     <choice minOccurs="0" maxOccurs="unbounded">  
387       <element ref="saml:Statement"/>
```

```

388         <element ref="saml:SubjectStatement"/>
389         <element ref="saml:AuthenticationStatement"/>
390         <element ref="saml:AuthorizationDecisionStatement"/>
391         <element ref="saml:AttributeStatement"/>
392     </choice>
393     <element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
394 </sequence>
395 <attribute name="MajorVersion" type="integer" use="required"/>
396 <attribute name="MinorVersion" type="integer" use="required"/>
397 <attribute name="AssertionID" type="saml:IDType" use="required"/>
398 <attribute name="Issuer" type="string" use="required"/>
399 <attribute name="IssueInstant" type="dateTime" use="required"/>
400 </complexType>

```

### 401 2.3.3.1. Element <Conditions>

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

- 406 • If any condition evaluates to Invalid, the assertion status is Invalid.
- 407 • If no condition evaluates to Invalid and one or more conditions evaluate to  
408 Indeterminate, the assertion status is Indeterminate.
- 409 • If no conditions are supplied or all the specified conditions evaluate to Valid, the assertion  
410 status is Valid.

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

414 The <Conditions> element contains the following element and attributes:

415 NotBefore [Optional]

416 Specifies the earliest time instant at which the assertion is valid.

417 NotOnOrAfter [Optional]

418 Specifies the time instant at which the assertion has expired.

419 <Condition> [Zero or more]

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

421 <AudienceRestrictionCondition> [Any Number]

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

423 <TargetRestrictionCondition> [Any Number]

424 The <TargetRestriction> condition is used to limit the use of the assertion to a particular  
425 relying party.

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

```

428 <element name="Conditions" type="saml:ConditionsType"/>
429 <complexType name="ConditionsType">
430     <choice minOccurs="0" maxOccurs="unbounded">
431         <element ref="saml:Condition"/>
432         <element ref="saml:AudienceRestrictionCondition"/>
433         <element ref="saml:TargetRestrictionCondition"/>
434     </choice>
435     <attribute name="NotBefore" type="dateTime" use="optional"/>
436     <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>

```

437 `</complexType>`

### 438 **2.3.3.1.1 Attributes *NotBefore* and *NotOnOrAfter***

439 The *NotBefore* and *NotOnOrAfter* attributes specify time limits on the validity of the assertion.

440 The *NotBefore* attribute specifies the time instant at which the validity interval begins. The  
441 *NotOnOrAfter* attribute specifies the time instant at which the validity interval has ended.

442 If the value for either *NotBefore* or *NotOnOrAfter* is omitted or is equal to the start of the epoch,  
443 it is considered unspecified. If the *NotBefore* attribute is unspecified (and if any other conditions  
444 that are supplied evaluate to *Valid*), the assertion is valid at any time before the time instant  
445 specified by the *NotOnOrAfter* attribute. If the *NotOnOrAfter* attribute is unspecified (and if any  
446 other conditions that are supplied evaluate to *Valid*), the assertion is valid from the time instant  
447 specified by the *NotBefore* attribute with no expiry. If neither attribute is specified (and if any other  
448 conditions that are supplied evaluate to *Valid*), the assertion is valid at any time.

449 The *NotBefore* and *NotOnOrAfter* attributes are defined to have the **dateTime** simple type that  
450 is built in to the W3C XML Schema Datatypes specification [**Schema2**]. All time instants are  
451 interpreted to be in Universal Coordinated Time (UTC) unless they explicitly indicate a time zone.

452 Implementations MUST NOT generate time instants that specify leap seconds.

### 453 **2.3.3.1.2 Element *<Condition>***

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

457 The following schema fragment defines the *<Condition>* element and its  
458 **ConditionAbstractType** complex type:

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

### 461 **2.3.3.1.3 Elements *<AudienceRestrictionCondition>* and *<Audience>***

462 The *<AudienceRestrictionCondition>* element specifies that the assertion is addressed to  
463 one or more specific audiences. Although a party that is outside the audiences specified is capable  
464 of drawing conclusions from an assertion, the issuer explicitly makes no representation as to  
465 accuracy or trustworthiness to such a party.

466 An audience is identified by a URI. The URI MAY identify a document that describes the terms and  
467 conditions of audience membership.

468 The condition evaluates to *Valid* if and only if the relying party is a member of one or more of the  
469 audiences specified.

470 The issuer of an assertion cannot prevent a party to whom it is disclosed from making a decision on  
471 the basis of the information provided. However, the *<AudienceRestrictionCondition>*  
472 element allows the issuer to state explicitly that no warranty is provided to such a party in a  
473 machine- and human-readable form. While there can be no guarantee that a court would upholding  
474 such a warranty exclusion in every circumstance, the probability of upholding the warranty  
475 exclusion is considerably improved.

476 The following schema fragment defines the *<AudienceRestrictionCondition>* element and  
477 its **AudienceRestrictionConditionType** complex type:

```
478 <element name="AudienceRestrictionCondition"  
479   type="saml:AudienceRestrictionConditionType"/>  
480 <complexType name="AudienceRestrictionConditionType">  
481   <complexContent>
```

```

482     <extension base="saml:ConditionAbstractType">
483       <sequence>
484         <element ref="saml:Audience"
485           minOccurs="1" maxOccurs="unbounded"/>
486       </sequence>
487     </extension>
488   </complexContent>
489 </complexType>
490 <element name="Audience" type="anyURI"/>

```

#### 491 2.3.3.1.4 Condition Type TargetRestrictionType

492 The <TargetRestriction> element is used to limit the use of the assertion to a particular relying  
493 party. This is useful to prevent malicious forwarding of assertions to unintended recipients.

494 The target is identified by a URI. The condition evaluates to true if one or more URIs identify the  
495 recipient or a resource managed by the recipient.

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

```

498 <element name="TargetRestrictionCondition"
499   type="saml:TargetRestrictionConditionType"/>
500 <complexType name="TargetRestrictionConditionType">
501   <complexContent>
502     <extension base="saml:ConditionAbstractType">
503       <sequence>
504         <element ref="saml:Target"
505           minOccurs="1" maxOccurs="unbounded"/>
506       </sequence>
507     </extension>
508   </complexContent>
509 </complexType>
510 <element name="Target" type="anyURI"/>

```

#### 511 2.3.3.2. Elements <Advice> and <AdviceElement>

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

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

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

- 519 • Include evidence supporting the assertion claims to be cited, either directly (through  
520 incorporating the claims) or indirectly (by reference to the supporting assertions).
- 521 • State a proof of the assertion claims.
- 522 • Specify the timing and distribution points for updates to the assertion.

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

```

525 <element name="Advice" type="saml:AdviceType"/>
526 <complexType name="AdviceType">
527   <sequence>
528     <choice minOccurs="0" maxOccurs="unbounded">
529       <element ref="saml:AssertionSpecifier"/>
530       <element ref="saml:AdviceElement"/>
531       <any namespace="##other" processContents="lax"/>
532     </choice>

```

```
533     </sequence>
534   </complexType>
535   <element name="AdviceElement" type="saml:AdviceAbstractType"/>
536   <complexType name="AdviceAbstractType"/>
```

## 537 2.4. Statements

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

### 539 2.4.1. Element <Statement>

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

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

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

### 547 2.4.2. Element <SubjectStatement>

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

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

```
555   <element name="SubjectStatement" type="saml:SubjectStatementAbstractType"/>
556   <complexType name="SubjectStatementAbstractType" abstract="true">
557     <complexContent>
558       <extension base="saml:StatementAbstractType">
559         <sequence>
560           <element ref="saml:Subject"/>
561         </sequence>
562       </extension>
563     </complexContent>
564   </complexType>
```

#### 565 2.4.2.1. Element <Subject>

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

568 <NameIdentifier>

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

570 <SubjectConfirmation>

571 Information that allows the subject to be authenticated.

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



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

```
579 <element name="Subject" type="saml:SubjectType"/>
580 <complexType name="SubjectType">
581 <choice maxOccurs="unbounded">
582 <sequence>
583 <element ref="saml:NameIdentifier"/>
584 <element ref="saml:SubjectConfirmation" minOccurs="0"/>
585 </sequence>
586 <element ref="saml:SubjectConfirmation"/>
587 </choice>
588 </complexType>
```

#### 589 2.4.2.2. Element <NameIdentifier>

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

592 SecurityDomain

593 The security domain governing the name of the subject.

594 Name

595 The name of the subject.

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

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

```
601 <element name="NameIdentifier" type="saml:NameIdentifierType"/>
602 <complexType name="NameIdentifierType">
603 <attribute name="SecurityDomain" type="string"/>
604 <attribute name="Name" type="string"/>
605 </complexType>
```

#### 606 2.4.2.3. Elements <SubjectConfirmation>, <ConfirmationMethod>, and 607 <SubjectConfirmationData>

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

610 <ConfirmationMethod> [One or more]

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

613 <SubjectConfirmationData> [Zero or more]

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

615 <ds:KeyInfo> [Optional]

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

618 The following schema fragment defines the <SubjectConfirmation> element and its  
619 **SubjectConfirmationType** complex type, along with the <SubjectConfirmationData>  
620 element and the <ConfirmationMethod> element:

```
621 <element name="SubjectConfirmation" type="saml:SubjectConfirmationType"/>
622 <complexType name="SubjectConfirmationType">
623 <sequence>
624 <element ref="saml:ConfirmationMethod" maxOccurs="unbounded"/>
625 <element ref="saml:SubjectConfirmationData" minOccurs="0"/>
```

```

626     <element ref="ds:KeyInfo" minOccurs="0"/>
627   </sequence>
628 </complexType>
629 <element name="SubjectConfirmationData" type="string" minOccurs="0"/>
630 <element name="ConfirmationMethod" type="anyURI"/>

```

### 631 2.4.3. Element <AuthenticationStatement>

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

636 AuthenticationMethod [Required]

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

639 AuthenticationInstant [Required]

640 Specifies the time at which the authentication took place.

641 <AuthenticationLocality> [Optional]

642 Specifies the DNS domain name and IP address for the system entity that was  
643 authenticated.

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

```

646 <element name="AuthenticationStatement"
647     type="saml:AuthenticationStatementType"/>
648 <complexType name="AuthenticationStatementType">
649   <complexContent>
650     <extension base="saml:SubjectStatementAbstractType">
651       <sequence>
652         <element ref="saml:AuthenticationLocality" minOccurs="0"/>
653         <element ref="saml:AuthorityBinding"
654             minOccurs="0" maxOccurs="unbounded"/>
655       </sequence>
656       <attribute name="AuthenticationMethod" type="anyURI"/>
657       <attribute name="AuthenticationInstant" type="dateTime"/>
658     </extension>
659   </complexContent>
660 </complexType>

```

#### 661 2.4.3.1. Element <AuthenticationLocality>

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

664 IPAddress [Optional]

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

666 DNSAddress [Required]

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

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

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

```

672 <element name="AuthenticationLocality"
673     type="saml:AuthenticationLocalityType"/>
674 <complexType name="AuthenticationLocalityType">
675   <attribute name="IPAddress" type="string" use="optional"/>

```

```
676     <attribute name="DNSAddress" type="string" use="optional"/>
677 </complexType>
```

### 678 2.4.3.2. Element <AuthorityBinding>

679 The <AuthorityBinding> element specifies the type of authority (authentication, attribute,  
680 authorization) that performed the authentication and points to it via URI:

681 AuthorityKind [Optional]

682 The type of authority that performed the authentication.

683 Binding [Optional]

684 The address of the authority.

685 The following schema fragment defines the <AuthorityBinding> element and its

686 **AuthorityBindingType** complex type and **AuthorityKindType** simple type:

```
687 <element name="AuthorityBinding" type="saml:AuthorityBindingType"/>
688 <complexType name="AuthorityBindingType">
689   <attribute name="AuthorityKind" type="saml:AuthorityKindType"/>
690   <attribute name="Binding" type="anyURI"/>
691 </complexType>
692 <simpleType name="AuthorityKindType">
693   <restriction base="string">
694     <enumeration value="authentication"/>
695     <enumeration value="attribute"/>
696     <enumeration value="authorization"/>
697   </restriction>
698 </simpleType>
```

### 699 2.4.4. Element <AuthorizationDecisionStatement>

700 The <AuthorizationDecisionStatement> element supplies a statement by the issuer that the  
701 request for access by the specified subject to the specified resource has resulted in the specified  
702 decision on the basis of some optionally specified evidence. It is of type  
703 **AuthorizationDecisionStatementType**, which extends **SubjectStatementAbstractType** with the  
704 addition of the following elements (in order) and attributes:

705 Resource [Optional]

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

707 Decision [Optional]

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

710 <Actions> [Required]

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

712 <Evidence> [Zero or more]

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

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

```
716 <element name="AuthorizationDecisionStatement"
717 type="saml:AuthorizationDecisionStatementType"/>
718 <complexType name="AuthorizationDecisionStatementType">
719   <complexContent>
720     <extension base="saml:SubjectStatementAbstractType">
721       <sequence>
722         <element ref="saml:Actions"/>
723         <element ref="saml:Evidence" minOccurs="0"
724           maxOccurs="unbounded"/>

```

```

725         </sequence>
726         <attribute name="Resource" type="anyURI" use="optional"/>
727         <attribute name="Decision" type="saml:DecisionType"
728             use="optional"/>
729     </extension>
730 </complexContent>
731 </complexType>

```

#### 732 2.4.4.1. Elements <Actions> and <Action>

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

735 Namespace [Optional]

736 A URI representing the namespace in which the names of specified actions are to be  
737 interpreted. If this element is absent, the namespace `http://www.oasis-`  
738 `open.org/committees/security/docs/draft-sstc-core-25/rwedc-negation` specified in section  
739 7.2.2 is in effect by default.

740 <Action> [One or more]

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

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

```

744     <element name="Actions" type="saml:ActionsType"/>
745     <complexType name="ActionsType">
746         <sequence>
747             <element ref="saml:Action" maxOccurs="unbounded"/>
748         </sequence>
749         <attribute name="Namespace" type="anyURI" use="optional"/>
750     </complexType>
751     <element name="Action" type="string"/>

```

#### 752 2.4.4.2. Element <Evidence>

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

755 The provision of an assertion as evidence MAY affect the reliance agreement between the client  
756 and the service. For example, in the case that the client presented an assertion to the service in a  
757 request, the service MAY use that assertion as evidence in making its response without endorsing  
758 the assertion as valid either to the client or any third party.

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

```

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

```

#### 761 2.4.5. Element <AttributeStatement>

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

765 <Attribute> [One or More]

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

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

```

769     <element name="AttributeStatement" type="saml:AttributeStatementType"/>
770     <complexType name="AttributeStatementType">
771         <complexContent>

```

```

772     <extension base="saml:SubjectStatementAbstractType">
773         <sequence>
774             <element ref="saml:Attribute" maxOccurs="unbounded"/>
775         </sequence>
776     </extension>
777 </complexContent>
778 </complexType>

```

#### 779 2.4.5.1. Elements <AttributeDesignator> and <Attribute>

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

784 AttributeNamespace [Required]

785 The namespace in which the AttributeName elements are interpreted.

786 AttributeName [Required]

787 The name of the attribute.

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

```

790 <element name="AttributeDesignator" type="saml:AttributeDesignatorType"/>
791 <complexType name="AttributeDesignatorType">
792     <attribute name="AttributeName" type="string"/>
793     <attribute name="AttributeNamespace" type="anyURI"/>
794 </complexType>

```

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

798 <AttributeValue> [Required]

799 The value of the attribute.

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

```

802 <element name="Attribute" type="saml:AttributeType"/>
803 <complexType name="AttributeType">
804     <complexContent>
805         <extension base="saml:AttributeDesignatorType">
806             <sequence>
807                 <element ref="saml:AttributeValue"/>
808             </sequence>
809         </extension>
810     </complexContent>
811 </complexType>

```

##### 812 2.4.5.1.1 Element <AttributeValue>

813 The <AttributeValue> element supplies the value of the specified attribute. It is of the  
814 **AttributeValueType** complex type, which allows the inclusion of any element in any namespace  
815 and specifies that lax schema validation is in effect.

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

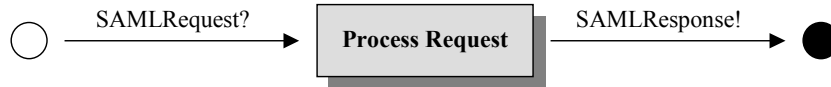
820 The following schema fragment defines the <AttributeValue> element and its  
821 **AttributeValueType** complex type:

```
822 <element name="AttributeValue" type="saml:AttributeValueType"/>
823 <complexType name="AttributeValueType">
824   <sequence>
825     <any namespace="##any" processContents="lax"
826       minOccurs="0" maxOccurs="unbounded"/>
827   </sequence>
828 </complexType>
```

## 829 3. SAML Protocol

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

833 SAML-aware clients MAY in addition use the SAML request-response protocol defined by the  
834 <Request> and <Response> elements. The client sends a <Request> element to a SAML  
835 service, and the service generates a <Response> element, as shown in Figure 2.



836

837

Figure 2: SAML Request-Response Protocol

### 838 3.1. Schema Header and Namespace Declarations

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

```
841 <schema  
842   targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-  
843   sstc-schema-protocol-25.xsd"  
844   xmlns="http://www.w3.org/2001/XMLSchema"  
845   xmlns:samlp="http://www.oasis-open.org/committees/security/docs/draft-sstc-  
846   schema-protocol-25.xsd"  
847   xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-  
848   schema-assertion-25.xsd"  
849   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
850   elementFormDefault="unqualified">  
851   <import namespace="http://www.oasis-open.org/committees/security/docs/draft-  
852   sstc-schema-assertion-25.xsd"  
853     schemaLocation="draft-sstc-schema-assertion-25.xsd"/>  
854   <import namespace="http://www.w3.org/2000/09/xmldsig#"  
855     schemaLocation="xmldsig-core-schema.xsd"/>  
856   <annotation>  
857     <documentation>draft-sstc-schema-protocol-25.xsd</documentation>  
858   </annotation>  
859   ...  
860 </schema>
```

861

### 862 3.2. Requests

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

#### 864 3.2.1. Complex Type RequestAbstractType

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

867 RequestID [Required]

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

871 MajorVersion [Required]  
 872       The major version of this request. The identifier for the version of SAML defined in this  
 873       specification is 1. Processing of this attribute is specified in Section 3.4.2.

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

877 <RespondWith> [Any Number]  
 878       Each <RespondWith> element specifies a type of response that is acceptable to the  
 879       requestor.

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

```
881 <complexType name="RequestAbstractType" abstract="true">
882   <sequence>
883     <element ref="samlp:RespondWith"
884       minOccurs="0" maxOccurs="unbounded"/>
885     <element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
886   </sequence>
887   <attribute name="RequestID" type="saml:IDType" use="required"/>
888   <attribute name="MajorVersion" type="integer" use="required"/>
889   <attribute name="MinorVersion" type="integer" use="required"/>
890 </complexType>
```

### 891 3.2.1.1. Element <RespondWith>

892 The <RespondWith> element specifies a type of response that is acceptable to the requestor. If  
 893 no <RespondWith> element is specified the default is SingleStatement. Acceptable values for  
 894 the <RespondWith> element are:

895 SingleStatement  
 896       An assertion carrying exactly one statement element.

897 MultipleStatement  
 898       An assertion carrying at least one statement element.

899 AuthenticationStatement  
 900       An assertion carrying an Authentication statement.

901 AuthorizationDecisionStatement  
 902       An assertion carrying an Authorization Decision statement.

903 AttributeStatement  
 904       An assertion carrying an Attribute statement.

905 Schema URI  
 906       An assertion containing additional elements from the specified schema.

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

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

### 909 3.2.2. Element <Request>

910 The <Request> element specifies a SAML request. It provides either a query or a request for a  
 911 specific assertion identified by <AssertionID> or <AssertionArtifact>. It has the complex  
 912 type **RequestType**, which extends **RequestAbstractType** by adding a choice of one of the  
 913 following elements:

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



916 <SubjectQuery>  
 917     An extension point that allows extension schemas to define new types of query that specify  
 918     a single SAML subject.

919 <AuthenticationQuery>  
 920     Makes a query for authentication information.

921 <AttributeQuery>  
 922     Makes a query for attribute information.

923 <AuthorizationDecisionQuery>  
 924     Makes a query for an authorization decision.

925 <AssertionID> [One or more]  
 926     Requests an assertion by reference to its assertion identifier.

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

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

```

931     <element name="Request" type="samlp:RequestType"/>
932     <complexType name="RequestType">
933       <complexContent>
934         <extension base="samlp:RequestAbstractType">
935           <choice>
936             <element ref="samlp:Query"/>
937             <element ref="samlp:SubjectQuery"/>
938             <element ref="samlp:AuthenticationQuery"/>
939             <element ref="samlp:AttributeQuery"/>
940             <element ref="samlp:AuthorizationDecisionQuery"/>
941             <element ref="saml:AssertionID" maxOccurs="unbounded"/>
942             <element ref="samlp:AssertionArtifact" maxOccurs="unbounded"/>
943           </choice>
944         </extension>
945       </complexContent>
946     </complexType>
947     <element name="AssertionArtifact" type="string"/>
  
```

## 948 3.3. Queries

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

### 950 3.3.1. Element <Query>

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

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

```

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

### 959 3.3.2. Element <SubjectQuery>

960 The <SubjectQuery> element is an extension point that allows new SAML queries that specify a  
 961 single SAML subject. Its **SubjectQueryAbstractType** complex type is abstract; extension elements

962 MUST use the `xsi:type` attribute to indicate the derived type. **SubjectQueryAbstractType** adds  
963 the `<Subject>` element.

964 The following schema fragment defines the `<SubjectQuery>` element and its  
965 **SubjectQueryAbstractType** complex type:

```
966 <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType"/>  
967 <complexType name="SubjectQueryAbstractType" abstract="true">  
968 <complexContent>  
969 <extension base="samlp:QueryAbstractType">  
970 <sequence>  
971 <element ref="saml:Subject"/>  
972 </sequence>  
973 </extension>  
974 </complexContent>  
975 </complexType>
```

### 976 3.3.3. Element `<AuthenticationQuery>`

977 The `<AuthenticationQuery>` element is used to make the query “What authentication  
978 assertions are available for this subject?” A successful response will be in the form of an assertion  
979 containing an authentication statement. This element is of type **AuthenticationQueryType**, which  
980 extends **SubjectQueryAbstractType** with the addition of the following element:

981 `<ConfirmationMethod>` [Optional]  
982 A filter for possible responses. If it is present, the query made is “What authentication  
983 assertions do you have for this subject with the supplied confirmation method?”

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

989 The following schema fragment defines the `<AuthenticationQuery>` type and its  
990 **AuthenticationQueryType** complex type:

```
991 <element name="AuthenticationQuery" type="samlp:AuthenticationQueryType"/>  
992 <complexType name="AuthenticationQueryType">  
993 <complexContent>  
994 <extension base="samlp:SubjectQueryAbstractType">  
995 <sequence>  
996 <element ref="saml:ConfirmationMethod" minOccurs="0"/>  
997 </sequence>  
998 </extension>  
999 </complexContent>  
1000 </complexType>
```

### 1001 3.3.4. Element `<AttributeQuery>`

1002 The `<AttributeQuery>` element is used to make the query “Return the requested attributes for  
1003 this subject.” The response will be in the form of an assertion containing an attribute statement.  
1004 This element is of type **AttributeQueryType**, which extends **SubjectQueryAbstractType** with the  
1005 addition of the following element and attribute:

1006 `<AttributeDesignator>` [Zero or more] (see Section 2.4.5.1)  
1007 Each `<AttributeDesignator>` element specifies an attribute whose value is to be  
1008 returned. If no attributes are specified, the list of desired attributes is implicit and  
1009 application-specific.

1010 The following schema fragment defines the `<AttributeQuery>` element and its  
1011 **AttributeQueryType** complex type:

```

1012 <element name="AttributeQuery" type="samlp:AttributeQueryType"/>
1013 <complexType name="AttributeQueryType">
1014   <complexContent>
1015     <extension base="samlp:SubjectQueryAbstractType">
1016       <sequence>
1017         <element ref="saml:AttributeDesignator"
1018           minOccurs="0" maxOccurs="unbounded"/>
1019       </sequence>
1020     </extension>
1021   </complexContent>
1022 </complexType>

```

### 1023 3.3.5. Element <AuthorizationDecisionQuery>

1024 The <AuthorizationDecisionQuery> element is used to make the query “Should these  
1025 actions on this resource be allowed for this subject, given this evidence?” The response will be in  
1026 the form of an assertion containing an authorization decision statement. This element is of type  
1027 **AuthorizationDecisionQueryType**, which extends **SubjectQueryAbstractType** with the addition  
1028 of the following elements and attribute:

1029 Resource [Required]

1030 A URI indicating the resource for which authorization is requested.

1031 <Actions> [Required]

1032 The actions for which authorization is requested.

1033 <Evidence> [Zero or more]

1034 An assertion that the responder MAY rely on in making its response.

1035 The following schema fragment defines the <AuthorizationDecisionQuery> element and its  
1036 **AuthorizationDecisionQueryType** complex type:

```

1037 <element name="AuthorizationDecisionQuery"
1038 type="samlp:AuthorizationDecisionQueryType"/>
1039 <complexType name="AuthorizationDecisionQueryType">
1040   <complexContent>
1041     <extension base="samlp:SubjectQueryAbstractType">
1042       <sequence>
1043         <element ref="saml:Actions"/>
1044         <element ref="saml:Evidence"
1045           minOccurs="0" maxOccurs="unbounded"/>
1046       </sequence>
1047       <attribute name="Resource" type="anyURI"/>
1048     </extension>
1049   </complexContent>
1050 </complexType>

```

## 1051 3.4. Responses

1052 The following sections define the SAML constructs that contain response information.

### 1053 3.4.1. Complex Type ResponseAbstractType

1054 All SAML responses are of types that are derived from the abstract **ResponseAbstractType**  
1055 complex type. This type defines common attributes that are associated with all SAML responses:

1056 ResponseID [Required]

1057 An identifier for the response. It is of type **IDType**, and MUST follow the requirements  
1058 specified by that type for identifier uniqueness.

- 1059 **InResponseTo** [Required]  
 1060     A reference to the identifier of the request to which the response corresponds. The value of  
 1061     this attribute **MUST** match the value of the corresponding `RequestID` attribute.
- 1062 **MajorVersion** [Required]  
 1063     The major version of this response. The identifier for the version of SAML defined in this  
 1064     specification is 1. Processing of this attribute is specified in Section 3.4.2.
- 1065 **MinorVersion** [Required]  
 1066     The minor version of this response. The identifier for the version of SAML defined in this  
 1067     specification is 0. Processing of this attribute is specified in Section 3.4.2.

1068 The following schema fragment defines the **ResponseAbstractType** complex type:

```

1069     <complexType name="ResponseAbstractType" abstract="true">
1070       <sequence>
1071         <element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
1072       </sequence>
1073       <attribute name="ResponseID" type="saml:IDType" use="required"/>
1074       <attribute name="InResponseTo" type="saml:IDType" use="required"/>
1075       <attribute name="MajorVersion" type="integer" use="required"/>
1076       <attribute name="MinorVersion" type="integer" use="required"/>
1077     </complexType>
  
```

### 1078 3.4.2. Element <Response>

1079 The <Response> element specifies the status of the corresponding SAML request and a list of  
 1080 zero or more assertions that answer the request. It has the complex type **ResponseType**, which  
 1081 extends **ResponseAbstractType** by adding the following elements (in an unbounded mixture) and  
 1082 attribute:

- 1083 <Status> [Required] (see Section 3.4.3)  
 1084     A code representing the status of the corresponding request.
- 1085 <Assertion> (see Section 2.3.3)  
 1086     Specifies an assertion by value.

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

```

1089     <element name="Response" type="samlp:ResponseType"/>
1090     <complexType name="ResponseType">
1091       <complexContent>
1092         <extension base="samlp:ResponseAbstractType">
1093           <sequence>
1094             <element ref="samlp:Status"/>
1095             <element ref="saml:Assertion"
1096               minOccurs="0" maxOccurs="unbounded"/>
1097           </sequence>
1098         </extension>
1099       </complexContent>
1100     </complexType>
  
```

### 1101 3.4.3. Element <Status>

1102 The <Status> element :

- 1103 <StatusCode> [Required]  
 1104     A code representing the status of the corresponding request.
- 1105 <StatusMessage> [Any Number]  
 1106     A message which **MAY** be returned to an operator.

1107 <StatusDetail> [Optional]  
1108 Specifies additional information concerning an error condition.

1109 The following schema fragment defines the <Status> element and its **StatusType** complex type:

```
1110 <element name="Status" type="samlp:StatusType"/>  
1111 <complexType name="StatusType">  
1112 <sequence>  
1113 <element ref="samlp:StatusCode"/>  
1114 <element ref="samlp:StatusMessage"  
1115 minOccurs="0" maxOccurs="unbounded"/>  
1116 <element ref="samlp:StatusDetail" minOccurs="0"/>  
1117 </sequence>  
1118 </complexType>
```

### 1119 3.4.3.1. Element <StatusCode>

1120 The <StatusCode> element specifies a code representing the status of the corresponding request  
1121 and an option sub code providing more specific information concerning a particular error status:

1122 Value [Required]  
1123 The status code value as defined below.

1124 <SubStatusCode> [Optional]  
1125 An optional subordinate status code value that provides more specific information on an  
1126 error condition.

1127 The following **StatusCode** values are defined:

1128 Success  
1129 The request succeeded.

1130 VersionMismatch  
1131 The receiver could not process the request because the version was incorrect.

1132 Reciever  
1133 The request could not be performed due to an error at the receiving end.

1134 Sender  
1135 The request could not be performed due to an error in the sender or in the request

1136 The following schema fragment defines the <StatusCode> element and its **StatusCodeType**  
1137 complex type and the **StatusCodeEnumType** simple type:

```
1138 <element name="StatusCode" type="samlp:StatusCodeType"/>  
1139 <complexType name="StatusCodeType">  
1140 <sequence>  
1141 <element ref="samlp:SubStatusCode" minOccurs="0"/>  
1142 </sequence>  
1143 <attribute name="Value" type="samlp:StatusCodeEnumType"/>  
1144 </complexType>  
1145 <simpleType name="StatusCodeEnumType">  
1146 <restriction base="QName">  
1147 <enumeration value="samlp:Success"/>  
1148 <enumeration value="samlp:VersionMismatch"/>  
1149 <enumeration value="samlp:Receiver"/>  
1150 <enumeration value="samlp:Sender"/>  
1151 </restriction>  
1152 </simpleType>
```

### 1153 3.4.3.2. Element <SubStatusCode>

1154 The <SubStatusCode> element specifies an additional code representing the status of the  
1155 corresponding request:

1156 Value [Required]  
 1157 The status code value as defined below.

1158 <SubStatusCode> [Optional]  
 1159 An optional subordinate status code value that provides an additional level of specific  
 1160 information on an error condition.

1161 The following **SubStatusCode** values are defined, additional codes MAY be defined in future  
 1162 versions of the SAML specification:

1163 RequestVersionTooHigh  
 1164 The protocol version specified in the request is a major upgrade from the highest protocol  
 1165 version supported by the responder.

1166 RequestVersionTooLow  
 1167 The responder cannot respond to the particular request using the SAML version specified  
 1168 in the request because it is too low.

1169 RequestVersionDeprecated  
 1170 The responder does not respond to any requests with the protocol version specified in the  
 1171 request.

1172 TooManyResponses  
 1173 The response would contain more elements than the responder will return.

1174 The following schema fragment defines the <SubStatusCode> element and its  
 1175 **SubStatusCodeType** complex type:

```
1176 <element name="SubStatusCode" type="samlp:SubStatusCodeType"/>
1177 <complexType name="SubStatusCodeType">
1178   <sequence>
1179     <element ref="samlp:SubStatusCode" minOccurs="0"/>
1180   </sequence>
1181   <attribute name="Value" type="QName"/>
1182 </complexType>
```

### 1183 3.4.3.3. Element <StatusMessage>

1184 The <StatusMessage> element specifies a message that MAY be returned to an operator:

1185 The following schema fragment defines the <StatusMessage> element and its  
 1186 **StatusMessageType** complex type:

```
1187 <element name="StatusMessage" type="string"/>
```

### 1188 3.4.3.4. Element <StatusDetail>

1189 The <StatusDetail> element MAY be used to specify additional information concerning an error  
 1190 condition.

1191 The following schema fragment defines the <StatusDetail> element and its **StatusDetailType**  
 1192 complex type:

```
1193 <element name="StatusDetail" type="samlp:StatusDetailType"/>
1194 <complexType name="StatusDetailType">
1195   <sequence>
1196     <any namespace="##any"
1197       processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1198   </sequence>
1199 </complexType>
```

### 1200 **3.4.4. Simple Type StatusCodeType**

1201 The **StatusCodeType** simple type is used in a response to specify the status of the request that  
1202 caused the response to be generated. The type enumerates the following possible values:

1203 Success

1204       The request succeeded.

1205 Failure

1206       The request could not be performed by the service.

1207 Error

1208       An error in the request prevented the service from processing it.

1209 Unknown

1210       The request failed for unknown reasons.

1211 The following schema fragment defines the **StatusCodeType** simple type:

```
1212 <simpleType name="StatusCodeType">  
1213   <restriction base="string">  
1214     <enumeration value="Success"/>  
1215     <enumeration value="Failure"/>  
1216     <enumeration value="Error"/>  
1217     <enumeration value="Unknown"/>  
1218   </restriction>  
1219 </simpleType>
```

## 4. SAML Versioning

1220

1221 SAML version information appears in the following elements:

- 1222 • <Assertion>
- 1223 • <Request>
- 1224 • <Response>

1225 The version numbering of the SAML assertion is independent of the version number of the SAML  
1226 request-response protocol. The version information for each consists of a major version number  
1227 and a minor version number, both of which are integers. In accordance with industry practice a  
1228 version number SHOULD be presented to the user in the form *Major.Minor*. This document defines  
1229 SAML Assertions 1.0 and SAML Protocol 1.0.

1230 The version number  $Major_B.Minor_B$  is higher than the version number  $Major_A.Minor_A$  if and only if:

1231  $Major_B > Major_A \vee ( ( Major_B = Major_A ) \wedge Minor_B = Minor_A )$

1232 Each revision of SAML SHALL assign version numbers to assertions, requests, and responses that  
1233 are the same as or higher than the corresponding version number in the SAML version that  
1234 immediately preceded it.

1235 New versions of SAML SHALL assign new version numbers as follows:

- 1236 • **Documentation change:**  $( Major_B = Major_A ) \wedge ( Minor_B = Minor_A )$   
1237 If the major and minor version numbers are unchanged, the new version *B* only introduces  
1238 changes to the documentation that raise no compatibility issues with an implementation of  
1239 version *A*.
- 1240 • **Minor upgrade:**  $( Major_B = Major_A ) \wedge ( Minor_B > Minor_A )$   
1241 If the major version number of versions *A* and *B* are the same and the minor version  
1242 number of *B* is higher than that of *A*, the new SAML version MAY introduce changes to the  
1243 SAML schema and semantics but any changes that are introduced in *B* SHALL be  
1244 compatible with version *A*.
- 1245 • **Major upgrade:**  $Major_B > Major_A$   
1246 If the major version of *B* number is higher than the major version of *A*, Version *B* MAY  
1247 introduce changes to the SAML schema and semantics that are incompatible with *A*.

### 4.1. Assertion Version

1248

1249 A SAML application MUST NOT issue any assertion whose version number is not supported.

1250 A SAML application MUST reject any assertion whose major version number is not supported.

1251 A SAML application MAY reject any assertion whose version number is higher than the highest  
1252 supported version.

### 4.2. Request Version

1253

1254 A SAML application SHOULD issue requests that specify the highest SAML version supported by  
1255 both the sender and recipient.

1256 If the SAML application does not know the capabilities of the recipient it should assume that it  
1257 supports the highest SAML version supported by the sender.



## 1258 **4.3. Response Version**

1259 A SAML application MUST NOT issue responses that specify a higher SAML version number than  
1260 the corresponding request.

1261 A SAML application MUST NOT issue a response that has a major version number that is lower  
1262 than the major version number of the corresponding request except to report the error

1263 `RequestVersionTooHigh`.

1264 Incompatible protocol versions MAY cause the following errors to be reported:

1265 `RequestVersionTooHigh`

1266       The protocol version specified in the request is a major upgrade from the highest protocol  
1267       version supported by the responder.

1268 `RequestVersionTooLow`

1269       The responder cannot respond to the particular request using the SAML version specified  
1270       in the request because it is too low.

1271 `RequestVersionDeprecated`

1272       The responder does not respond to any requests with the protocol version specified in the  
1273       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.

Note :

- SAML documents may be the subject of signatures from in many different packaging contexts. [SIG] 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: 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 a secure channel and the AP has authenticated to the RP.

Request/Response messages: the originator has authenticated to the destination and the destination has obtained the assertion directly from the originator (no intermediaries) thru 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.

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.

### 5.2. Request/Response Signing

All SAML requests and responses MAY be signed using the XML Signature. This is reflected in the schema – Section 3.3.1 & 3.5.1.

1317 **5.3. Signature Inheritance (a.k.a. super-signatures & sub-**  
1318 **messages)**

1319 **5.3.1. Rationale**

1320 SAML assertions may be embedded within request or response messages or other XML  
1321 messages, which may be signed. Request or response messages may themselves be contained  
1322 within other messages that are based on other XML messaging frameworks (e.g., SOAP) and the  
1323 composite object may be the subject of a signature. Another possibility is that SAML assertions or  
1324 request/response messages are embedded within a non-XML messaging object (e.g., MIME  
1325 package) and signed.

1326 In such a case, the SAML sub-message (Assertion, request, response) may be viewed as inheriting  
1327 a signature from the "super-signature" over the enclosing object, provided certain constraints are  
1328 met.

1329 (1) An assertion may be viewed as inheriting a signature from a super signature, if the super  
1330 signature applies all the elements within the assertion.

1331 A SAML request or response may be viewed as inheriting a signature from a super signature, if the  
1332 super signature applies to all the elements within the response.

1333 **5.3.2. Rules for SAML Signature Inheritance**

1334 Signature inheritance: occurs when SAML message (assertion/request/response) is not signed but  
1335 is enclosed within signed SAML such that the signature applies to all of the elements within the  
1336 message. In such a case, the SAML message is said to inherit the signature and may be  
1337 considered equivalent to the case where it is explicitly signed. The SAML message inherits the  
1338 "closest enclosing signature".

1339 But if SAML messages need to be passed around by themselves, or embedded in other messages,  
1340 they would need to be signed as per section 2.1

1341 **5.4. XML Signature Profile**

1342 The XML Signature [**XMLSig**] specification calls out a general XML syntax for signing data with  
1343 many flexibilities and choices. This section details the constraints on these facilities so that SAML  
1344 processors do not have to deal with the full generality of ML Signature processing.

1345 **5.4.1. Signing formats**

1346 XML Signature has three ways of representing signature in a document viz: enveloping, enveloped  
1347 and detached.

1348 SAML assertions and protocols MUST use the enveloped signatures for signing assertions.

1349 **5.4.2. CanonicalizationMethod**

1350 XML Signature REQUIRES the Canonical XML (omits comments)  
1351 (<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>). SAML implementations SHOULD use  
1352 Canonical XML with no comments.

1353 **5.4.3. Transforms**

1354 [Sig] REQUIRES the enveloped signature transform  
1355 <http://www.w3.org/2000/09/xmldsig#enveloped-signature>

1356 **5.4.4. KeyInfo**

1357 SAML does not restrict or impose any restrictions in this area. Therefore following [SIG] keyInfo  
1358 may be absent.

1359 **5.4.5. Binding between statements in a multi-statement assertion**

1360 Use of signing does not affect semantics of statements within assertions in any way, as stated in  
1361 this document Sections 1 thru 4.

1362 **5.4.6. Security considerations**

1363 **5.4.6.1. Replay Attack**

1364 The mechanisms stated here-in does not offer any counter measures against a replay attack. Other  
1365 mechanisms like sequence numbers, time stamps, expiration et al need to be explored to prevent a  
1366 replay attack.

## 1367 **6. SAML Extensions**

1368 The SAML schemas support extensibility. An example of an application that extends SAML  
1369 assertions is the XTAML system for management of embedded trust roots [XTAML]. The following  
1370 sections explain how to use the extensibility features in SAML to create extension schemas.

1371 Note that elements in the SAML schemas are not blocked from substitution, so that all SAML  
1372 elements MAY serve as the head element of a substitution group. Also, types are not defined as  
1373 *final*, so that all SAML types MAY be extended and restricted. The following sections discuss  
1374 only elements that have been specifically designed to support extensibility.

### 1375 **6.1. Assertion Schema Extension**

1376 The SAML assertion schema is designed to permit separate processing of the assertion package  
1377 and the statements it contains, if the extension mechanism is used for either part.

1378 The following elements are intended specifically for use as extension points in an extension  
1379 schema; their types are set to *abstract*, so that the use of an `xsi:type` attribute with these  
1380 elements is REQUIRED:

- 1381 • `<Assertion>`
- 1382 • `<Condition>`
- 1383 • `<Statement>`
- 1384 • `<SubjectStatement>`
- 1385 • `<AdviceElement>`

1386 In addition, the following elements that are directly usable as part of SAML MAY be extended:

- 1387 • `<AuthenticationStatement>`
- 1388 • `<AuthorizationDecisionStatement>`
- 1389 • `<AttributeStatement>`
- 1390 • `<AudienceRestrictionCondition>`

1391 Finally, the following elements are defined to allow elements from arbitrary namespaces within  
1392 them, which serves as a built-in extension point without requiring an extension schema:

- 1393 • `<AttributeValue>`
- 1394 • `<Advice>`

### 1395 **6.2. Protocol Schema Extension**

1396 The following elements are intended specifically for use as extension points in an extension  
1397 schema; their types are set to *abstract*, so that the use of an `xsi:type` attribute with these  
1398 elements is REQUIRED:

- 1399 • `<Query>`
- 1400 • `<SubjectQuery>`

1401 In addition, the following elements that are directly usable as part of SAML MAY be extended:

- 1402 • `<Request>`

- 1403 • <AuthenticationQuery>
- 1404 • <AuthorizationDecisionQuery>
- 1405 • <AttributeQuery>
- 1406 • <Response>

### 1407 6.3. Use of Type Derivation and Substitution Groups

1408 W3C XML Schema [Schema1] provides two principal mechanisms for specifying an element of an  
1409 extended type: type derivation and substitution groups.

1410 For example, a <Statement> element can be assigned the type **NewStatementType** by means of  
1411 the `xsi:type` attribute. For such an element to be schema-valid, **NewStatementType** needs to be  
1412 derived from **StatementType**. The following example of a SAML assertion assumes that the  
1413 extension schema (represented by the `new:` prefix) has defined this new type:

```
1414 <saml:Assertion ...>  
1415   <saml:Statement xsi:type="new:NewStatementType">  
1416     ...  
1417   </saml:Statement>  
1418 </saml:Assertion>
```

1419 Alternatively, the extension schema can define a <NewStatement> element that is a member of a  
1420 substitution group that has <Statement> as a head element. For the substituted element to be  
1421 schema-valid, it needs to have a type that matches or is derived from the head element's type. The  
1422 following is an example of an extension schema fragment that defines this new element:

```
1423 <xsd:element "NewStatement" type="new:NewStatementType"  
1424   substitutionGroup="saml:Statement"/>
```

1425 The substitution group declaration allows the <NewStatement> element to be used anywhere the  
1426 SAML <Statement> element can be used. The following is an example of a SAML assertion that  
1427 uses the extension element:

```
1428 <saml:Assertion ...>  
1429   <new:NewStatement>  
1430     ...  
1431   </new:NewStatement>  
1432 </saml:Assertion>
```

1433 The choice of extension method has no effect on the semantics of the XML document but does  
1434 have implications for interoperability.

1435 The advantages of type derivation are as follows:

- 1436 • A document can be more fully interpreted by a parser that does not have access to the  
1437 extension schema because a "native" SAML element is available.
- 1438 • At the time of writing, some W3C XML Schema validators do not support substitution  
1439 groups, whereas the `xsi:type` attribute is widely supported.

1440 The advantage of substitution groups is that a document can be explained without the need to  
1441 explain the functioning of the `xsi:type` attribute.

## 1442 **7. SAML-Defined Identifiers**

1443 The following sections define URI-based identifiers for common authentication protocols and  
1444 actions.

1445 Where possible an existing URN is used to specify a protocol. In the case of IETF protocols the  
1446 URN of the most current RFC that specifies the protocol is used. URIs created specifically for  
1447 SAML have the initial stem:

1448 <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/>

### 1449 **7.1. Confirmation Method Identifiers**

1450 The following identifiers MAY be used in the <ConfirmationMethod> element (see Section  
1451 2.4.2.3) to refer to common authentication protocols.

#### 1452 **7.1.1. SAML Artifact:**

1453 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/artifact>

1454 <SubjectConfirmationData>: *Base64 ( Artifact )*

1455 The subject of the assertion is the party that can present the SAML Artifact value specified in  
1456 <SubjectConfirmationData>.

#### 1457 **7.1.2. SAML Artifact (SHA-1):**

1458 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/artifact-sha1>

1459 <SubjectConfirmationData>: *Base64 ( SHA1 ( Artifact ) )*

1460 The subject of the assertion is the party that can present a SAML Artifact such that the SHA1 digest  
1461 of the specified artifact matches the value specified in <SubjectConfirmationData>.

#### 1462 **7.1.3. Holder of Key:**

1463 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/Holder-Of-Key>

1464 <ds:KeyInfo>: Any cryptographic key

1465 The subject of the assertion is the party that can demonstrate that it is the holder of the private  
1466 component of the key specified in <ds:KeyInfo>.

#### 1467 **7.1.4. Sender Vouches:**

1468 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/sender-vouches>

1469 Indicates that no other information is available about the context of use of the assertion. The  
1470 Relying party SHOULD utilize other means to determine if it should process the assertion further.

#### 1471 **7.1.5. Password (Pass-Through):**

1472 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/password>

1473 <SubjectConfirmationData>: *Base64 ( Password )*

1474 The subject of the assertion is the party that can present the password value specified in  
1475 <SubjectConfirmationData>.

1476 The username of the subject is specified by means of the <NameIdentifier> element.

### 1477 **7.1.6. Password (One-Way-Function SHA-1):**

1478 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/password-sha1>

1479 <SubjectConfirmationData>: *Base64 ( SHA1 ( Password ) )*

1480 The subject of the assertion is the party that can present the password such that the SHA1 digest of  
1481 the specified password matches the value specified in <SubjectConfirmationData>.

1482 The username of the subject is specified by means of the <NameIdentifier> element.

### 1483 **7.1.7. Kerberos**

1484 **URI:** urn:ietf:rfc:1510

1485 <SubjectConfirmationData>: A Kerberos Ticket

1486 The subject is authenticated by means of the Kerberos protocol [RFC 1510], an instantiation of the  
1487 Needham-Schroeder symmetric key authentication mechanism [Needham78].

### 1488 **7.1.8. SSL/TLS Certificate Based Client Authentication:**

1489 **URI:** urn:ietf:rfc:2246

1490 <ds:KeyInfo>: Any cryptographic key

### 1491 **7.1.9. Object Authenticator (SHA-1):**

1492 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/object-sha1>

1493 <SubjectConfirmationData>: *Base64 ( SHA1 ( Object ) )*

1494 This authenticator element is the result of computing a digest, using the SHA-1 hash algorithm. It is  
1495 used when the subject can be represented as a binary string, for example when it is an XML  
1496 document or the disk image of executable code. Any preprocessing of the subject prior to  
1497 computation of the digest is out of scope. The name of the subject should be conveyed in an  
1498 accompanying NameIdentifier element.

### 1499 **7.1.10. PKCS#7**

1500 **URI:** urn:ietf:rfc:2315

1501 <SubjectConfirmationData>: *Base64 ( PKCS#7 ( Object ) )*

1502 This authenticator element is signed data in PKCS#7 format [PKCS#7]. The posited identity of the  
1503 signer must be conveyed in an accompanying NameIdentifier element. This subject type may be  
1504 included in the subject field of an authentication query, in which case the corresponding response  
1505 indicates whether the posited signer is, indeed, the signer. It may be included in an attribute query,  
1506 in which case, the requested attribute values for the subject authenticated by the signed data are  
1507 returned. It may be included in an authorization query, in which case, the access request  
1508 represented by the signed data shall be identified by the accompanying object element, and the



1509 corresponding authorization decision assertion indicates whether the signer is authorized for the  
1510 access request represented by the object element.

### 1511 **7.1.11. Cryptographic Message Syntax**

1512 **URI:** urn:ietf:rfc:2630

1513 <SubjectConfirmationData>: *Base64* ( CMS ( *Object* ) )

1514 This authenticator element is signed data in CMS format [CMS]. See also 7.1.10

### 1515 **7.1.12. XML Digital Signature**

1516 **URI:** urn:ietf:rfc:2630

1517 <SubjectConfirmationData>: *Base64* ( XML-SIG ( *Object* ) )

1518 <ds:KeyInfo>: A cryptographic signing key

1519 This authenticator element is signed data in XML Signature format. See also 7.1.10

## 1520 **7.2. Action Namespace Identifiers**

1521 The following identifiers MAY be used in the `ActionNamespace` attribute (see Section 2.4.4.1) to  
1522 refer to common sets of actions to perform on resources.

### 1523 **7.2.1. Read/Write/Execute/Delete/Control:**

1524 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/rwedc>

1525 Defined actions:

1526 `Read Write Execute Delete Control`

1527 These actions are interpreted in the normal manner, i.e.

1528 `Read`

1529 `The subject may read the resource`

1530 `Write`

1531 `The subject may modify the resource`

1532 `Execute`

1533 `The subject may execute the resource`

1534 `Delete`

1535 `The subject may delete the resource`

1536 `Control`

1537 `The subject may specify the access control policy for the resource`

### 1538 **7.2.2. Read/Write/Execute/Delete/Control with Negation:**

1539 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/rwedc-negation>

1540 Defined actions:

1541 `Read Write Execute Delete Control ~Read ~Write ~Execute ~Delete ~Control`

1542 The actions specified in section 7.2.1 are interpreted in the same manner described there. Actions  
1543 prefixed with a tilde ~ are negated permissions and are used to affirmatively specify that the stated

1544 permission is denied. Thus a subject described as being authorized to perform the action `~Read` is  
1545 affirmatively denied read permission.

1546 An application MUST NOT authorize both an action and its negated form.

### 1547 **7.2.3. Get/Head/Put/Post:**

1548 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/ghpp>

1549 Defined actions:

1550 GET HEAD PUT POST

1551 These actions bind to the corresponding HTTP operations. For example a subject authorized to  
1552 perform the GET action on a resource is authorized to retrieve it.

1553 The GET and HEAD actions loosely correspond to the conventional read permission and the PUT  
1554 and POST actions to the write permission. The correspondence is not exact however since a HTTP  
1555 GET operation may cause data to be modified and a POST operation may cause modification to a  
1556 resource other than the one specified in the request. For this reason a separate Action URI  
1557 specifier is provided.

### 1558 **7.2.4. UNIX File Permissions:**

1559 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/unix>

1560 The defined actions are the set of UNIX file access permissions expressed in the numeric (octal)  
1561 notation.

1562 The action string is a four digit numeric code:

1563 *extended user group world*

1564 Where the *extended* access permission has the value

1565 +2 if sgid is set

1566 +4 if suid is set

1567 The *user group* and *world* access permissions have the value

1568 +1 if execute permission is granted

1569 +2 if write permission is granted

1570 +4 if read permission is granted

1571 For example `0754` denotes the UNIX file access permission: user read, write and execute, group  
1572 read and execute and world read.

## 8. SAML Schema Listings

1573

1574 The following sections contain complete listings of the assertion and protocol schemas for SAML.

### 8.1. Assertion Schema

1575

1576 Following is a complete listing of the SAML assertion schema [SAML-XSD].

```
1577 <?xml version="1.0" encoding="UTF-8"?>
1578 <!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-Baker
1579 (VeriSign Inc.) -->
1580 <schema
1581     targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-
1582     sstc-schema-assertion-25.xsd"
1583     xmlns="http://www.w3.org/2001/XMLSchema" xmlns:saml="http://www.oasis-
1584     open.org/committees/security/docs/draft-sstc-schema-assertion-25.xsd"
1585     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1586     elementFormDefault="unqualified">
1587     <import namespace="http://www.w3.org/2000/09/xmldsig#"
1588         schemaLocation="xmldsig-core-schema.xsd"/>
1589     <annotation>
1590         <documentation>draft-sstc-schema-assertion-25.xsd</documentation>
1591     </annotation>
1592     <simpleType name="IDType">
1593         <restriction base="string"/>
1594     </simpleType>
1595     <simpleType name="DecisionType">
1596         <restriction base="string">
1597             <enumeration value="Permit"/>
1598             <enumeration value="Deny"/>
1599             <enumeration value="Indeterminate"/>
1600         </restriction>
1601     </simpleType>
1602     <element name="AssertionSpecifier" type="saml:AssertionSpecifierType"/>
1603     <complexType name="AssertionSpecifierType">
1604         <choice>
1605             <element ref="saml:AssertionID"/>
1606             <element ref="saml:Assertion"/>
1607         </choice>
1608     </complexType>
1609     <element name="AssertionID" type="saml:IDType"/>
1610     <element name="Assertion" type="saml:AssertionType"/>
1611     <complexType name="AssertionType">
1612         <sequence>
1613             <element ref="saml:Conditions" minOccurs="0"/>
1614             <element ref="saml:Advice" minOccurs="0"/>
1615             <choice minOccurs="0" maxOccurs="unbounded">
1616                 <element ref="saml:Statement"/>
1617                 <element ref="saml:SubjectStatement"/>
1618                 <element ref="saml:AuthenticationStatement"/>
1619                 <element ref="saml:AuthorizationDecisionStatement"/>
1620                 <element ref="saml:AttributeStatement"/>
1621             </choice>
1622             <element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
1623         </sequence>
1624         <attribute name="MajorVersion" type="integer" use="required"/>
1625         <attribute name="MinorVersion" type="integer" use="required"/>
1626         <attribute name="AssertionID" type="saml:IDType" use="required"/>
1627         <attribute name="Issuer" type="string" use="required"/>
1628         <attribute name="IssueInstant" type="dateTime" use="required"/>
1629     </complexType>
```

```

1630 <element name="Conditions" type="saml:ConditionsType"/>
1631 <complexType name="ConditionsType">
1632   <choice minOccurs="0" maxOccurs="unbounded">
1633     <element ref="saml:Condition"/>
1634     <element ref="saml:AudienceRestrictionCondition"/>
1635   </choice>
1636   <attribute name="NotBefore" type="dateTime" use="optional"/>
1637   <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>
1638 </complexType>
1639 <element name="Condition" type="saml:ConditionAbstractType"/>
1640 <complexType name="ConditionAbstractType" abstract="true"/>
1641 <element name="AudienceRestrictionCondition"
1642   type="saml:AudienceRestrictionConditionType"/>
1643 <complexType name="AudienceRestrictionConditionType">
1644   <complexContent>
1645     <extension base="saml:ConditionAbstractType">
1646       <sequence>
1647         <element ref="saml:Audience" maxOccurs="unbounded"/>
1648       </sequence>
1649     </extension>
1650   </complexContent>
1651 </complexType>
1652 <element name="Audience" type="anyURI"/>
1653 <element name="TargetRestrictionCondition"
1654   type="saml:TargetRestrictionConditionType"/>
1655 <complexType name="TargetRestrictionConditionType">
1656   <complexContent>
1657     <extension base="saml:ConditionAbstractType">
1658       <sequence>
1659         <element ref="saml:Target"
1660           minOccurs="1" maxOccurs="unbounded"/>
1661       </sequence>
1662     </extension>
1663   </complexContent>
1664 </complexType>
1665 <element name="Target" type="anyURI"/>
1666 <element name="Advice" type="saml:AdviceType"/>
1667 <complexType name="AdviceType">
1668   <sequence>
1669     <choice minOccurs="0" maxOccurs="unbounded">
1670       <element ref="saml:AssertionSpecifier"/>
1671       <element ref="saml:AdviceElement"/>
1672       <any namespace="##other" processContents="lax"/>
1673     </choice>
1674   </sequence>
1675 </complexType>
1676 <element name="AdviceElement" type="saml:AdviceAbstractType"/>
1677 <complexType name="AdviceAbstractType"/>
1678 <element name="Statement" type="saml:StatementAbstractType"/>
1679 <complexType name="StatementAbstractType" abstract="true"/>
1680 <element name="SubjectStatement" type="saml:SubjectStatementAbstractType"/>
1681 <complexType name="SubjectStatementAbstractType" abstract="true">
1682   <complexContent>
1683     <extension base="saml:StatementAbstractType">
1684       <sequence>
1685         <element ref="saml:Subject"/>
1686       </sequence>
1687     </extension>
1688   </complexContent>
1689 </complexType>
1690 <element name="Subject" type="saml:SubjectType"/>
1691 <complexType name="SubjectType">
1692   <choice maxOccurs="unbounded">

```

```

1693     <sequence>
1694         <element ref="saml:NameIdentifier"/>
1695         <element ref="saml:SubjectConfirmation" minOccurs="0"/>
1696     </sequence>
1697     <element ref="saml:SubjectConfirmation"/>
1698 </choice>
1699 </complexType>
1700 <element name="NameIdentifier" type="saml:NameIdentifierType"/>
1701 <complexType name="NameIdentifierType">
1702     <attribute name="SecurityDomain" type="string"/>
1703     <attribute name="Name" type="string"/>
1704 </complexType>
1705 <element name="SubjectConfirmation" type="saml:SubjectConfirmationType"/>
1706 <complexType name="SubjectConfirmationType">
1707     <sequence>
1708         <element ref="saml:ConfirmationMethod" maxOccurs="unbounded"/>
1709         <element ref="saml:SubjectConfirmationData" minOccurs="0"/>
1710         <element ref="ds:KeyInfo" minOccurs="0"/>
1711     </sequence>
1712 </complexType>
1713 <element name="SubjectConfirmationData" type="string" minOccurs="0"/>
1714 <element name="ConfirmationMethod" type="anyURI"/>
1715 <element name="AuthenticationStatement"
1716     type="saml:AuthenticationStatementType"/>
1717 <complexType name="AuthenticationStatementType">
1718     <complexContent>
1719         <extension base="saml:SubjectStatementAbstractType">
1720             <sequence>
1721                 <element ref="saml:AuthenticationLocality" minOccurs="0"/>
1722                 <element ref="saml:AuthorityBinding"
1723                     minOccurs="0" maxOccurs="unbounded"/>
1724             </sequence>
1725             <attribute name="AuthenticationMethod" type="anyURI"/>
1726             <attribute name="AuthenticationInstant" type="dateTime"/>
1727         </extension>
1728     </complexContent>
1729 </complexType>
1730 <element name="AuthenticationLocality"
1731     type="saml:AuthenticationLocalityType"/>
1732 <complexType name="AuthenticationLocalityType">
1733     <attribute name="IPAddress" type="string" use="optional"/>
1734     <attribute name="DNSAddress" type="string" use="optional"/>
1735 </complexType>
1736 <element name="AuthorityBinding" type="saml:AuthorityBindingType"/>
1737 <complexType name="AuthorityBindingType">
1738     <attribute name="AuthorityKind" type="saml:AuthorityKindType"/>
1739     <attribute name="Binding" type="anyURI"/>
1740 </complexType>
1741 <simpleType name="AuthorityKindType">
1742     <restriction base="string">
1743         <enumeration value="authentication"/>
1744         <enumeration value="attribute"/>
1745         <enumeration value="authorization"/>
1746     </restriction>
1747 </simpleType>
1748 <element name="AuthorizationDecisionStatement"
1749     type="saml:AuthorizationDecisionStatementType"/>
1750 <complexType name="AuthorizationDecisionStatementType">
1751     <complexContent>
1752         <extension base="saml:SubjectStatementAbstractType">
1753             <sequence>
1754                 <element ref="saml:Actions"/>
1755                 <element ref="saml:Evidence"

```

```

1756             minOccurs="0" maxOccurs="unbounded"/>
1757         </sequence>
1758         <attribute name="Resource" type="anyURI" use="optional"/>
1759         <attribute name="Decision"
1760             type="saml:DecisionType" use="optional"/>
1761     </extension>
1762 </complexContent>
1763 </complexType>
1764 <element name="Actions" type="saml:ActionsType"/>
1765 <complexType name="ActionsType">
1766     <sequence>
1767         <element ref="saml:Action" maxOccurs="unbounded"/>
1768     </sequence>
1769     <attribute name="Namespace" type="anyURI" use="optional"/>
1770 </complexType>
1771 <element name="Action" type="string"/>
1772 <element name="Evidence" type="saml:AssertionSpecifierType"/>
1773 <element name="AttributeStatement" type="saml:AttributeStatementType"/>
1774 <complexType name="AttributeStatementType">
1775     <complexContent>
1776         <extension base="saml:SubjectStatementAbstractType">
1777             <sequence>
1778                 <element ref="saml:Attribute" maxOccurs="unbounded"/>
1779             </sequence>
1780         </extension>
1781     </complexContent>
1782 </complexType>
1783 <element name="AttributeDesignator" type="saml:AttributeDesignatorType"/>
1784 <complexType name="AttributeDesignatorType">
1785     <attribute name="AttributeName" type="string"/>
1786     <attribute name="AttributeNamespace" type="anyURI"/>
1787 </complexType>
1788 <element name="Attribute" type="saml:AttributeType"/>
1789 <complexType name="AttributeType">
1790     <complexContent>
1791         <extension base="saml:AttributeDesignatorType">
1792             <sequence>
1793                 <element ref="saml:AttributeValue"/>
1794             </sequence>
1795         </extension>
1796     </complexContent>
1797 </complexType>
1798 <element name="AttributeValue" type="saml:AttributeValueType"/>
1799 <complexType name="AttributeValueType">
1800     <sequence>
1801         <any namespace="##any" processContents="lax"
1802             minOccurs="0" maxOccurs="unbounded"/>
1803     </sequence>
1804 </complexType>
1805 </schema>
1806

```

## 1807 8.2. Protocol Schema

1808 Following is a complete listing of the SAML protocol schema [SAML-XSD].

```

1809 <?xml version="1.0" encoding="UTF-8"?>
1810 <!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-Baker
1811 (VeriSign Inc.) -->
1812 <schema
1813     targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-
1814     sstc-schema-protocol-25.xsd"
1815     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"

```

```

1816     xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-
1817 schema-assertion-25.xsd"
1818     xmlns:samlp="http://www.oasis-open.org/committees/security/docs/draft-sstc-
1819 schema-protocol-25.xsd"
1820     xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
1821     <import
1822         namespace="http://www.oasis-open.org/committees/security/docs/draft-sstc-
1823 schema-assertion-25.xsd"
1824         schemaLocation="draft-sstc-schema-assertion-25.xsd"/>
1825     <import namespace="http://www.w3.org/2000/09/xmlsig#"
1826         schemaLocation="xmlsig-core-schema.xsd"/>
1827     <annotation>
1828         <documentation>draft-sstc-schema-protocol-25.xsd</documentation>
1829     </annotation>
1830     <complexType name="RequestAbstractType" abstract="true">
1831         <sequence>
1832             <element ref="samlp:RespondWith"
1833                 minOccurs="0" maxOccurs="unbounded"/>
1834             <element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
1835         </sequence>
1836         <attribute name="RequestID" type="saml:IDType" use="required"/>
1837         <attribute name="MajorVersion" type="integer" use="required"/>
1838         <attribute name="MinorVersion" type="integer" use="required"/>
1839     </complexType>
1840     <element name="RespondWith" type="anyURI"/>
1841     <element name="Request" type="samlp:RequestType"/>
1842     <complexType name="RequestType">
1843         <complexContent>
1844             <extension base="samlp:RequestAbstractType">
1845                 <choice>
1846                     <element ref="samlp:Query"/>
1847                     <element ref="samlp:SubjectQuery"/>
1848                     <element ref="samlp:AuthenticationQuery"/>
1849                     <element ref="samlp:AttributeQuery"/>
1850                     <element ref="samlp:AuthorizationDecisionQuery"/>
1851                     <element ref="saml:AssertionID" maxOccurs="unbounded"/>
1852                     <element ref="samlp:AssertionArtifact" maxOccurs="unbounded"/>
1853                 </choice>
1854             </extension>
1855         </complexContent>
1856     </complexType>
1857     <element name="AssertionArtifact" type="string"/>
1858     <element name="Query" type="samlp:QueryAbstractType"/>
1859     <complexType name="QueryAbstractType" abstract="true"/>
1860     <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType"/>
1861     <complexType name="SubjectQueryAbstractType" abstract="true">
1862         <complexContent>
1863             <extension base="samlp:QueryAbstractType">
1864                 <sequence>
1865                     <element ref="saml:Subject"/>
1866                 </sequence>
1867             </extension>
1868         </complexContent>
1869     </complexType>
1870     <element name="AuthenticationQuery" type="samlp:AuthenticationQueryType"/>
1871     <complexType name="AuthenticationQueryType">
1872         <complexContent>
1873             <extension base="samlp:SubjectQueryAbstractType">
1874                 <sequence>
1875                     <element ref="saml:ConfirmationMethod" minOccurs="0"/>
1876                 </sequence>
1877             </extension>
1878         </complexContent>

```

```

1879 </complexType>
1880 <element name="AttributeQuery" type="samlp:AttributeQueryType"/>
1881 <complexType name="AttributeQueryType">
1882   <complexContent>
1883     <extension base="samlp:SubjectQueryAbstractType">
1884       <sequence>
1885         <element ref="saml:AttributeDesignator"
1886           minOccurs="0" maxOccurs="unbounded"/>
1887       </sequence>
1888     </extension>
1889   </complexContent>
1890 </complexType>
1891 <element name="AuthorizationDecisionQuery"
1892   type="samlp:AuthorizationDecisionQueryType"/>
1893 <complexType name="AuthorizationDecisionQueryType">
1894   <complexContent>
1895     <extension base="samlp:SubjectQueryAbstractType">
1896       <sequence>
1897         <element ref="saml:Actions"/>
1898         <element ref="saml:Evidence"
1899           minOccurs="0" maxOccurs="unbounded"/>
1900       </sequence>
1901       <attribute name="Resource" type="anyURI"/>
1902     </extension>
1903   </complexContent>
1904 </complexType>
1905 <complexType name="ResponseAbstractType" abstract="true">
1906   <sequence>
1907     <element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
1908   </sequence>
1909   <attribute name="ResponseID" type="saml:IDType" use="required"/>
1910   <attribute name="InResponseTo" type="saml:IDType" use="required"/>
1911   <attribute name="MajorVersion" type="integer" use="required"/>
1912   <attribute name="MinorVersion" type="integer" use="required"/>
1913 </complexType>
1914
1915 <element name="Response" type="samlp:ResponseType"/>
1916 <complexType name="ResponseType">
1917   <complexContent>
1918     <extension base="samlp:ResponseAbstractType">
1919       <sequence>
1920         <element ref="samlp:Status"/>
1921         <element ref="saml:Assertion"
1922           minOccurs="0" maxOccurs="unbounded"/>
1923       </sequence>
1924     </extension>
1925   </complexContent>
1926 </complexType>
1927 <element name="Status" type="samlp:StatusType"/>
1928 <complexType name="StatusType">
1929   <sequence>
1930     <element ref="samlp:StatusCode"/>
1931     <element ref="samlp:StatusMessage"
1932       minOccurs="0" maxOccurs="unbounded"/>
1933     <element ref="samlp:StatusDetail" minOccurs="0"/>
1934   </sequence>
1935 </complexType>
1936 <element name="StatusCode" type="samlp:StatusCodeType"/>
1937 <complexType name="StatusCodeType">
1938   <sequence>
1939     <element ref="samlp:SubStatusCode" minOccurs="0"/>
1940   </sequence>
1941   <attribute name="Value" type="samlp:StatusCodeEnumType"/>

```



```
1942 </complexType>
1943 <simpleType name="StatusCodeEnumType">
1944   <restriction base="QName">
1945     <enumeration value="samlp:Success"/>
1946     <enumeration value="samlp:VersionMismatch"/>
1947     <enumeration value="samlp:Receiver"/>
1948     <enumeration value="samlp:Sender"/>
1949   </restriction>
1950 </simpleType>
1951 <element name="SubStatusCode" type="samlp:SubStatusCodeType"/>
1952 <complexType name="SubStatusCodeType">
1953   <sequence>
1954     <element ref="samlp:SubStatusCode" minOccurs="0"/>
1955   </sequence>
1956   <attribute name="Value" type="QName"/>
1957 </complexType>
1958 <element name="StatusMessage" type="string"/>
1959 <element name="StatusDetail" type="samlp:StatusDetailType"/>
1960 <complexType name="StatusDetailType">
1961   <sequence>
1962     <any namespace="##any"
1963       processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1964   </sequence>
1965 </complexType>
1966 </schema>
1967
```

## 9. References

1968

- 1969 [Needham78] R. Needham et al., *Using Encryption for Authentication in Large Networks of Computers*, Communications of the ACM, Vol. 21 (12), pp. 993-999, December 1978.
- 1970
- 1971
- 1972 [Kern-84] B. Kernighan, Rob Pike *The UNIX Programming Environment*, (March 1984) Prentice Hall Computer Books;
- 1973
- 1974 [PKCS1] B. Kaliski, *PKCS #1: RSA Encryption Version 2.0*, RSA Laboratories, also IETF RFC 2437, October 1998. <http://www.ietf.org/rfc/rfc2437.txt>
- 1975
- 1976 [PKCS7] B. Kaliski., "PKCS #7: Cryptographic Message Syntax, Version 1.5.", RFC 2315, March 1998.
- 1977
- 1978 [RFC 1510] J. Kohl, C. Neuman. *The Kerberos Network Authentication Service (V5)*. September 1993. <http://www.ietf.org/rfc/rfc1510.txt>
- 1979
- 1980 [RFC 2246] T. Dierks, C. Allen. *The TLS Protocol Version 1.0*. January 1999. <http://www.ietf.org/rfc/rfc2246.txt>
- 1981
- 1982 [RFC 2630] R. Housley. Cryptographic Message Syntax. June 1999. <http://www.ietf.org/rfc/rfc630.txt>
- 1983
- 1984 [RFC 2648] R. Moats. *A URN Namespace for IETF Documents*. August 1999. <http://www.ietf.org/rfc/rfc2648.txt>
- 1985
- 1986 [RFC 3075] D. Eastlake, J. Reagle, D. Solo. XML-Signature Syntax and Processing. March 2001. <http://www.ietf.org/rfc/rfc3075.txt>
- 1987
- 1988 [RFC2104] H. Krawczyk et al., *HMAC: Keyed Hashing for Message Authentication*, <http://www.ietf.org/rfc/rfc2104.txt>, IETF RFC 2104, February 1997.
- 1989
- 1990 [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997
- 1991
- 1992 [SAMLBind] P. Mishra et al., *Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML)*, <http://www.oasis-open.org/committees/security/docs/draft-sstc-bindings-model-07.pdf>, OASIS, December 2001.
- 1993
- 1994
- 1995
- 1996 [SAMLConform] **TBS**
- 1997 [SAMLGloss] J. Hodges et al., *Glossary for the OASIS Security Assertion Markup Language (SAML)*, <http://www.oasis-open.org/committees/security/docs/draft-sstc-glossary-02.pdf>, OASIS, December 2001.
- 1998
- 1999
- 2000
- 2001 [SAMLXSD] P. Hallam-Baker et al., *SAML protocol schema*, <http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-21.xsd>, OASIS, December 2001.
- 2002
- 2003
- 2004 [SAMLSecure] **TBS**
- 2005 [SAMLXSD] P. Hallam-Baker et al., *SAML assertion schema*, <http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-21.xsd>, OASIS, December 2001.
- 2006
- 2007
- 2008 [Schema1] H. S. Thompson et al., *XML Schema Part 1: Structures*, <http://www.w3.org/TR/xmlschema-1/>, World Wide Web Consortium Recommendation, May 2001.
- 2009
- 2010
- 2011 [Schema2] P. V. Biron et al., *XML Schema Part 2: Datatypes*, <http://www.w3.org/TR/xmlschema-2/>, World Wide Web Consortium Recommendation, May 2001.
- 2012
- 2013
- 2014 [XMLEnc] *XML Encryption Specification*, In development.

2015	<b>[XMLSig]</b>	D. Eastlake et al., <i>XML-Signature Syntax and Processing</i> , <a href="http://www.w3.org/TR/xmlsig-core/">http://www.w3.org/TR/xmlsig-core/</a> , World Wide Web Consortium.
2016		
2017	<b>[XMLSig-XSD]</b>	XML Signature Schema available from <a href="http://www.w3.org/TR/2000/CR-xmlsig-core-20001031/xmlsig-core-schema.xsd">http://www.w3.org/TR/2000/CR-xmlsig-core-20001031/xmlsig-core-schema.xsd</a> .
2018		
2019	<b>[XTAML]</b>	P. Hallam-Baker, <i>XML Trust Axiom Markup Language 1.0</i> , <a href="http://www.xmltrustcenter.org/">http://www.xmltrustcenter.org/</a> , VeriSign Inc. September 2001.
2020		

2021

## Appendix A. Notices

2022 OASIS takes no position regarding the validity or scope of any intellectual property or other rights  
2023 that might be claimed to pertain to the implementation or use of the technology described in this  
2024 document or the extent to which any license under such rights might or might not be available;  
2025 neither does it represent that it has made any effort to identify any such rights. Information on  
2026 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS  
2027 website. Copies of claims of rights made available for publication and any assurances of licenses to  
2028 be made available, or the result of an attempt made to obtain a general license or permission for  
2029 the use of such proprietary rights by implementors or users of this specification, can be obtained  
2030 from the OASIS Executive Director.

2031 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
2032 applications, or other proprietary rights which may cover technology that may be required to  
2033 implement this specification. Please address the information to the OASIS Executive Director.

2034 Copyright © The Organization for the Advancement of Structured Information Standards [OASIS]  
2035 2001. All Rights Reserved.

2036 This document and translations of it may be copied and furnished to others, and derivative works  
2037 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
2038 published and distributed, in whole or in part, without restriction of any kind, provided that the above  
2039 copyright notice and this paragraph are included on all such copies and derivative works. However,  
2040 this document itself may not be modified in any way, such as by removing the copyright notice or  
2041 references to OASIS, except as needed for the purpose of developing OASIS specifications, in  
2042 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights  
2043 document must be followed, or as required to translate it into languages other than English.

2044 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
2045 successors or assigns.

2046 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
2047 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
2048 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY  
2049 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
2050 PARTICULAR PURPOSE.

Page: 6

[PHB1] This will need to be edited before final release

Page: 7

[PHB2][TBD: XML examples of SAML subjects]

Page: 7

[PHB3][TBD: XML examples of SAML assertions]

Page: 7

[PHB4][TBD: XML examples of requests and responses]

Page: 9

[PHB5][TBD: XML examples of extension]

Page: 10

[PHB6]Update with final name spaces