



ACCELERATING ELECTRONIC BUSINESS

1

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

2 Document identifier: draft-sstc-core-[2726](#)

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

4 Publication date: [February 14th 2002](#)~~January 10th 2002~~

5 Maturity Level: Committee Working Draft

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

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

8 Editors:

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

11 Contributors:

12 Carlisle Adams, Entrust
13 Scott Cantor, The Ohio State University
14 Marc Chanliau, Netegrity
15 Nigel Edwards, Hewlett-Packard
16 Marlena Erdos, Tivoli
17 Stephen Farrell, Baltimore Technologies
18 Simon Godik, Crosslogic
19 Jeff Hodges, Oblix
20 Charles Knouse, Oblix
21 [Hal Lockhart, Entegrity](#)
22 Chris McLaren, Netegrity
23 Prateek Mishra, Netegrity
24 RL "Bob" Morgan, University of Washington
25 Tim Moses, Entrust
26 David Orchard, BEA
27 Joe Pato, Hewlett Packard
28 Darren Platt, RSA Security
29 Irving Reid, Baltimore[Technologies](#)
30 Krishna Sankar, Cisco Systems Inc

31

32

33

34

35

36

36		
37	ASSERTIONS AND PROTOCOL FOR THE OASIS SECURITY ASSERTION MARKUP	
38	LANGUAGE (SAML)	1
39	1. INTRODUCTION	<u>610</u>
40	1.1. NOTATION	<u>610</u>
41	1.2. SCHEMA ORGANIZATION AND NAMESPACES	<u>610</u>
42	1.2.1. Time Values.	<u>744</u>
43	1.2.2. Comparing SAML values	<u>744</u>
44	1.3. SAML CONCEPTS (NON-NORMATIVE)	<u>744</u>
45	1.3.1. Overview	<u>744</u>
46	1.3.2. SAML and URI-Based Identifiers	<u>943</u>
47	1.3.3. SAML and Extensibility	<u>943</u>
48	2. SAML ASSERTIONS	<u>1014</u>
49	2.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS	<u>1014</u>
50	2.2. SIMPLE TYPES	<u>1014</u>
51	2.2.1. Simple Types IDType and IDReferenceType	<u>1014</u>
52	2.2.2. Simple Type DecisionType	<u>1145</u>
53	2.3. ASSERTIONS	<u>1145</u>
54	2.3.1. Element <AssertionSpecifier>	<u>1145</u>
55	2.3.2. Element <AssertionID>	<u>1246</u>
56	2.3.3. Element <Assertion>	<u>1246</u>
57	2.3.3.1. Element <Conditions>	<u>1347</u>
58	2.3.3.1.1 Attributes NotBefore and NotOnOrAfter	<u>1418</u>
59	2.3.3.1.2 Element <Condition>	<u>1448</u>
60	2.3.3.1.3 Elements <AudienceRestrictionCondition> and <Audience>	<u>1448</u>
61	2.3.3.1.4 Elements <TargetRestrictionCondition> and <Target>	<u>1549</u>
62	2.3.3.2. Elements <Advice> and <AdviceElement>	<u>1549</u>
63	2.4. STATEMENTS	<u>1620</u>

64	2.4.1. <i>Element <Statement></i>	1620
65	2.4.2. <i>Element <SubjectStatement></i>	1620
66	2.4.2.1. Element <Subject>	1721
67	2.4.2.2. Element <NameIdentifier>	1721
68	2.4.2.3. Elements <SubjectConfirmation>, <ConfirmationMethod>, and <SubjectConfirmationData>	1824
69	2.4.3. <i>Element <AuthenticationStatement></i>	1822
70	2.4.3.1. Element <AuthenticationLocality>	1923
71	2.4.3.2. Element <AuthorityBinding>	1923
72	2.4.4. <i>Element <AuthorizationDecisionStatement></i>	2024
73	2.4.4.1. Elements <Actions> and <Action>	2125
74	2.4.4.2. Element <Evidence>	2125
75	2.4.5. <i>Element <AttributeStatement></i>	2226
76	2.4.5.1. Elements <AttributeDesignator> and <Attribute>	2226
77	2.4.5.1.1. Element <AttributeValue>	2327
78	3. SAML PROTOCOL	2428
79	3.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS	2428
80	3.2. REQUESTS	2428
81	3.2.1. <i>Complex Type RequestAbstractType</i>	2428
82	3.2.1.1. Element <RespondWith>	2529
83	3.2.2. <i>Element <Request></i>	2630
84	3.2.3. <i>Element <AssertionArtifact></i>	2730
85	3.3. QUERIES	2734
86	3.3.1. <i>Element <Query></i>	2734
87	3.3.2. <i>Element <SubjectQuery></i>	2734
88	3.3.3. <i>Element <AuthenticationQuery></i>	2734
89	3.3.4. <i>Element <AttributeQuery></i>	2832
90	3.3.5. <i>Element <AuthorizationDecisionQuery></i>	2832
91	3.4. RESPONSES	2933
92	3.4.1. <i>Complex Type ResponseAbstractType</i>	2933
93	3.4.2. <i>Element <Response></i>	3034

94	3.4.3. <i>Element <Status></i>	<u>3034</u>
95	3.4.3.1. Element <StatusCode>	<u>3134</u>
96	3.4.3.2. Element <SubStatusCode>	<u>3135</u>
97	3.4.3.3. Element <StatusMessage>	<u>3236</u>
98	3.4.3.4. Element <StatusDetail>	<u>3236</u>
99	3.4.4. <i>Responses to <AuthenticationQuery> and <AttributeQuery></i>	<u>3236</u>
100	4. SAML VERSIONING	<u>3437</u>
101	4.1. ASSERTION VERSION	<u>3437</u>
102	4.2. REQUEST VERSION	<u>3437</u>
103	4.3. RESPONSE VERSION	<u>3538</u>
104	5. SAML & XML-SIGNATURE SYNTAX AND PROCESSING	<u>3639</u>
105	5.1. SIGNING ASSERTIONS	<u>3639</u>
106	5.2. REQUEST/RESPONSE SIGNING	<u>3740</u>
107	5.3. SIGNATURE INHERITANCE	<u>3740</u>
108	5.3.1. <i>Rationale</i>	<u>3740</u>
109	5.3.2. <i>Rules for SAML Signature Inheritance</i>	<u>3740</u>
110	5.4. XML SIGNATURE PROFILE	<u>3740</u>
111	5.4.1. <i>Signing formats</i>	<u>3740</u>
112	5.4.2. <i>CanonicalizationMethod</i>	<u>3740</u>
113	5.4.3. <i>Transforms</i>	<u>3844</u>
114	5.4.4. <i>KeyInfo</i>	<u>3844</u>
115	5.4.5. <i>Binding between statements in a multi-statement assertion</i>	<u>3844</u>
116	6. SAML EXTENSIONS	<u>3942</u>
117	6.1. ASSERTION SCHEMA EXTENSION	<u>3942</u>
118	6.2. PROTOCOL SCHEMA EXTENSION	<u>3942</u>
119	6.3. USE OF TYPE DERIVATION AND SUBSTITUTION GROUPS	<u>4043</u>

120	7. SAML-DEFINED IDENTIFIERS	<u>4144</u>
121	7.1. CONFIRMATION METHOD IDENTIFIERS	<u>4144</u>
122	7.1.1. SAML Artifact:	<u>4144</u>
123	7.1.2. SAML Artifact (SHA-1):	<u>4144</u>
124	7.1.3. Holder of Key:	<u>4144</u>
125	7.1.4. Sender Vouches:	<u>4144</u>
126	7.1.5. Password (Pass-Through):	<u>4144</u>
127	7.1.6. Password (One-Way-Function SHA-1):	<u>4245</u>
128	7.1.7. Kerberos	<u>4245</u>
129	7.1.8. SSL/TLS Certificate Based Client Authentication:	<u>4245</u>
130	7.1.9. Object Authenticator (SHA-1):	<u>4245</u>
131	7.1.10. PKCS#7	<u>4245</u>
132	7.1.11. Cryptographic Message Syntax	<u>4346</u>
133	7.1.12. XML Digital Signature	<u>4346</u>
134	7.2. ACTION NAMESPACE IDENTIFIERS	<u>4346</u>
135	7.2.1. Read/Write/Execute/Delete/Control:	<u>4346</u>
136	7.2.2. Read/Write/Execute/Delete/Control with Negation:	<u>4346</u>
137	7.2.3. Get/Head/Put/Post:	<u>4447</u>
138	7.2.4. UNIX File Permissions:	<u>4447</u>
139	8. SAML SCHEMA LISTINGS	<u>4548</u>
140	8.1. ASSERTION SCHEMA	<u>4548</u>
141	8.2. PROTOCOL SCHEMA	<u>4851</u>
142	9. REFERENCES	<u>5255</u>
143	APPENDIX A. NOTICES	<u>5457</u>
144		

1. Introduction

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

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

1.1. Notation

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

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

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

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

Listings of SAML schemas appear like this.

Example code listings appear like this.

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

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

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

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

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

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

1.2. Schema Organization and Namespaces

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

<http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-2726.xsd>

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

<http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-2726.xsd>

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

188 The assertion schema is imported into the protocol schema. Also imported into both schemas is the
189 schema for XML Signature **[XMLSig-XSD]**, which is associated with the following XML namespace:
190 <http://www.w3.org/2000/09/xmldsig#>

191 **1.2.1. Time Values.**

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

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

196 **1.2.2. Comparing SAML values**

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

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

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

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

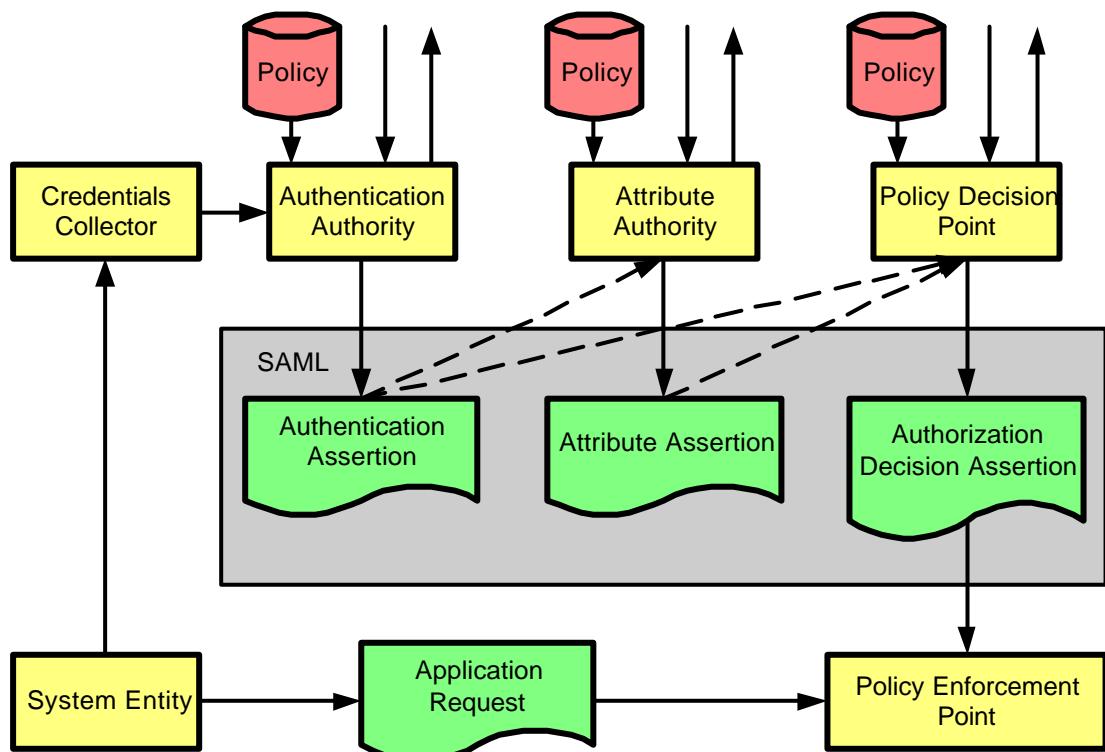
218 **1.3. SAML Concepts (Non-Normative)**

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

222 **1.3.1. Overview**

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

228 Assertions can convey information about authentication acts performed by subjects, attributes of
 229 subjects, and authorization decisions about whether subjects are allowed to access certain
 230 resources. Assertions are represented as XML constructs and have a nested structure, whereby a
 231 single assertion might contain several different internal statements about authentication,
 232 authorization, and attributes. Note that authentication assertions merely describe acts of
 233 authentication that happened previously.
 234 Assertions are issued by SAML authorities, namely, authentication authorities, attribute authorities,
 235 and policy decision points. SAML defines a protocol by which clients can request assertions from
 236 SAML authorities and get a response from them. This protocol, consisting of XML-based request
 237 and response message formats, can be bound to many different underlying communications and
 238 transport protocols; SAML currently defines one binding, to SOAP over HTTP.
 239 SAML authorities can use various sources of information, such as external policy stores and
 240 assertions that were received as input in requests, in creating their responses. Thus, while clients
 241 always consume assertions, SAML authorities can be both producers and consumers of assertions.
 242 The following model is conceptual only; for example, it does not account for real-world information
 243 flow or the possibility of combining of authorities into a single system.



244
 245 **Figure 1 The SAML Domain Model**
 246 One major design goal for SAML is Single Sign-On (SSO), the ability of a user to authenticate in
 247 one domain and use resources in other domains without re-authenticating. However, SAML can be
 248 used in various configurations to support additional scenarios as well. Several profiles of SAML are
 249 defined that support different styles of SSO and the securing of SOAP payloads.
 250 The assertion and protocol data formats are defined in this specification. The bindings and profiles
 251 are defined in a separate specification [[SAMLBind](#)]. A conformance program for SAML is defined
 252 in the conformance specification [[SAMLConform](#)]. Security issues are discussed in a separate
 253 security and privacy considerations specification [[SAMLSecure](#)].

254 **1.3.2. SAML and URI-Based Identifiers**

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

258 **urn:ietf:rfc:1510**

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

261 **<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#rwedc>**

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

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

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

273 **1.3.3. SAML and Extensibility**

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

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

278 2. SAML Assertions

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

- 281 ?? **Authentication**: The specified subject was authenticated by a particular means at a
282 particular time.
- 283 ?? **Authorization Decision**: A request to allow the specified subject to access the specified
284 resource has been granted or denied.
- 285 ?? **Attribute**: The specified subject is associated with the supplied attributes.

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

289 2.1. Schema Header and Namespace Declarations

290 The following schema fragment defines the XML namespaces and other header information for the
291 assertion schema:

```
292 <schema
293   targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-
294 | sstc-schema-assertion-2726.xsd"
295   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
296   xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-
297 | schema-assertion-2726.xsd"
298   xmlns="http://www.w3.org/2001/XMLSchema"
299   elementFormDefault="unqualified">
300     <import namespace="http://www.w3.org/2000/09/xmldsig#"
301       schemaLocation="xmldsig-core-schema.xsd"/>
302     <annotation>
303       <documentation>draft-sstc-schema-assertion-2726.xsd</documentation>
304     </annotation>
305   ...
306 </schema>
```

307 2.2. Simple Types

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

309 2.2.1. Simple Types **IDType** and **IDReferenceType**

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

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

- 313 ?? Any party that assigns an identifier MUST ensure that there is negligible probability that that
314 party or any other party will accidentally assign the same identifier to a different data object.
- 315 ?? Where a data object declares that it has a particular identifier, there MUST be exactly one
316 such declaration.

317 The mechanism by which the application ensures that the identifier is unique is left to the
318 implementation. In the case that a pseudorandom technique is employed, the probability of two
319 randomly chosen identifiers being identical MUST be less than 2^{-128} and SHOULD be less than
320 2^{-160} . This requirement MAY be met by applying Base64 encoding to a randomly chosen value 128
321 or 160 bits in length.

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

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

```
326     <simpleType name="IDType">
327         <restriction base="string" />
328     </simpleType>
329     <simpleType name="IDReferenceType">
330         <restriction base="string" />
331     </simpleType>
```

332 2.2.2. Simple Type **DecisionType**

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

335 Permit

336 The specified action is permitted.

337 Deny

338 The specified action is denied.

339 Indeterminate

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

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

```
342     <simpleType name="DecisionType">
343         <restriction base="string" >
344             <enumeration value="Permit" />
345             <enumeration value="Deny" />
346             <enumeration value="Indeterminate" />
347         </restriction>
348     </simpleType>
```

349 2.3. Assertions

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

351 2.3.1. Element **<AssertionSpecifier>**

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

354 **<AssertionIDReference>**

355 Specifies an assertion by reference to the value of the assertion's **AssertionID** attribute.

356 **<Assertion>**

357 Specifies an assertion by value.

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

```
360     <element name="AssertionSpecifier" type="saml:AssertionSpecifierType" />
361     <complexType name="AssertionSpecifierType" >
362         <choice>
363             <element ref ="saml:AssertionIDReference" />
364             <element ref ="saml:Assertion" />
365         </choice>
366     </complexType>
```

367 **2.3.2. Element <AssertionID>**

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

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

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

372 **2.3.3. Element <Assertion>**

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

375 MajorVersion [Required]

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

378 MinorVersion [Required]

379 The minor version of this assertion. The identifier for the version of SAML defined in this
380 specification is 0. Processing of this attribute is specified in Section 3.4.4.

381 AssertionID [Required]

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

384 Issuer [Required]

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

388 IssueInstant [Required]

389 The time instant of issue in UTC as described in section 1.2.1.

390 <Conditions> [Optional]

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

392 <Advice> [Optional]

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

395 <Signature> [Optional]

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

397 One or more of the following statement elements:

398 <Statement>

399 A statement defined in an extension schema.

400 <SubjectStatement>

401 A subject statement defined in an extension schema.

402 <AuthenticationStatement>

403 An authentication statement.

404 <AuthorizationDecisionStatement>

405 An authorization decision statement.

406 <AttributeStatement>

407 An attribute statement.

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

```

410 <element name="Assertion" type="saml:AssertionType" />
411 <complexType name="AssertionType">
412   <sequence>
413     <element ref="saml:Conditions" minOccurs="0" />
414     <element ref="saml:Advice" minOccurs="0" />
415     <choice maxOccurs="unbounded">
416       <element ref="saml:Statement" />
417       <element ref="saml:SubjectStatement" />
418       <element ref="saml:AuthenticationStatement" />
419       <element ref="saml:AuthorizationDecisionStatement" />
420       <element ref="saml:AttributeStatement" />
421     </choice>
422     <element ref="ds:Signature" minOccurs="0" />
423   </sequence>
424   <attribute name="MajorVersion" type="integer" use="required" />
425   <attribute name="MinorVersion" type="integer" use="required" />
426   <attribute name="AssertionID" type="saml:IDType" use="required" />
427   <attribute name="Issuer" type="string" use="required" />
428   <attribute name="IssueInstant" type="dateTime" use="required" />
429 </complexType>
```

430 2.3.3.1. Element <Conditions>

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

- 435 ?? If any condition evaluates to Invalid, the assertion status is Invalid.
- 436 ?? If no condition evaluates to Invalid and one or more conditions evaluate to
 437 Indeterminate, the assertion status is Indeterminate.
- 438 ?? If no conditions are supplied or all the specified conditions evaluate to Valid, the assertion
 439 status is Valid.

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

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

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

- 446 NotBefore [Optional]
 447 Specifies the earliest time instant at which the assertion is valid. The time value is encoded
 448 in UTC as described in section 1.2.1.
- 449 NotOnOrAfter [Optional]
 450 Specifies the time instant at which the assertion has expired. The time value is encoded in
 451 UTC as described in section 1.2.1.
- 452 <Condition> [Any Number]
 453 Provides an extension point allowing extension schemas to define new conditions.
- 454 <AudienceRestrictionCondition> [Any Number]
 455 Specifies that the assertion is addressed to a particular audience.
- 456 <TargetRestrictionCondition> [Any Number]
 457 The <TargetRestriction> condition is used to limit the use of the assertion to a particular
 458 relying party.

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

```
461 <element name="Conditions" type="saml:ConditionsType" />
462 <complexType name="ConditionsType">
463   <choice minOccurs="0" maxOccurs="unbounded">
464     <element ref="saml:Condition" />
465     <element ref="saml:AudienceRestrictionCondition" />
466     <element ref="saml:TargetRestrictionCondition" />
467   </choice>
468   <attribute name="NotBefore" type="dateTime" use="optional"/>
469   <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>
470 </complexType>
```

471 2.3.3.1.1 Attributes *NotBefore* and *NotOnOrAfter*

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

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

475 If the value for either `NotBefore` or `NotOnOrAfter` is omitted ~~or is equal to the start of the epoch,~~
476 it is considered unspecified. If the `NotBefore` attribute is unspecified (and if any other conditions
477 that are supplied evaluate to `Valid`), the assertion is valid at any time before the time instant
478 specified by the `NotOnOrAfter` attribute. If the `NotOnOrAfter` attribute is unspecified (and if any
479 other conditions that are supplied evaluate to `Valid`), the assertion is valid from the time instant
480 specified by the `NotBefore` attribute with no expiry. If neither attribute is specified (and if any other
481 conditions that are supplied evaluate to `Valid`), the assertion is valid at any time.

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

487 2.3.3.1.2 Element *<Condition>*

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

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

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

495 2.3.3.1.3 Elements *<AudienceRestrictionCondition>* and *<Audience>*

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

501 `<Audience>`
502 A URI that identifies an intended audience. The URI MAY identify a document that
503 describes the terms and conditions of audience membership.

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

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

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

```
514 <element name="AudienceRestrictionCondition"
515   type="saml:AudienceRestrictionConditionType" />
516 <complexType name="AudienceRestrictionConditionType">
517   <complexContent>
518     <extension base="saml:ConditionAbstractType">
519       <sequence>
520         <element ref="saml:Audience" maxOccurs="unbounded" />
521       </sequence>
522     </extension>
523   </complexContent>
524 </complexType>
525 <element name="Audience" type="anyURI" />
```

526 2.3.3.1.4 Elements `<TargetRestrictionCondition>` and `<Target>`

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

530 <Target>
531 A URI that identifies an intended relying party.

532 The `TargetRestrictionCondition` evaluates to `Valid` if and only if one or more URIs identify the
533 recipient or a resource managed by the recipient.

534 The following schema fragment defines the `<TargetRestrictionCondition>` element and its
535 **TargetRestrictionConditionType** complex type:

```
536 <element name="TargetRestrictionCondition"
537   type="saml:TargetRestrictionConditionType" />
538 <complexType name="TargetRestrictionConditionType">
539   <complexContent>
540     <extension base="saml:ConditionAbstractType">
541       <sequence>
542         <element ref="saml:Target"
543           minOccurs="1" maxOccurs="unbounded" />
544       </sequence>
545     </extension>
546   </complexContent>
547 </complexType>
548 <element name="Target" type="anyURI" />
```

549 2.3.3.2. Elements `<Advice>` and `<AdviceElement>`

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

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

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

557 ?? Include evidence supporting the assertion claims to be cited, either directly (through
558 incorporating the claims) or indirectly (by reference to the supporting assertions).
559 ?? State a proof of the assertion claims.
560 ?? Specify the timing and distribution points for updates to the assertion.

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

```
563 <element name="Advice" type="saml:AdviceType" />
564 <complexType name="AdviceType">
565   <sequence>
566     <choice minOccurs="0" maxOccurs="unbounded">
567       <element ref="saml:AssertionSpecifier" />
568       <element ref="saml:AdviceElement" />
569       <any namespace="###other" processContents="lax" />
570     </choice>
571   </sequence>
572 </complexType>
573 <element name="AdviceElement" type="saml:AdviceAbstractType" />
574 <complexType name="AdviceAbstractType">
```

575 2.4. Statements

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

577 2.4.1. Element **<Statement>**

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

581 The following schema fragment defines the `<Statement>` element and its
582 **StatementAbstractType** complex type:

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

585 2.4.2. Element **<SubjectStatement>**

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

591 The following schema fragment defines the `<SubjectStatement>` element and its
592 **SubjectStatementAbstractType** abstract type:

```
593 <element name="SubjectStatement" type="saml:SubjectStatementAbstractType" />
594 <complexType name="SubjectStatementAbstractType" abstract="true" >
595   <complexContent>
596     <extension base="saml:StatementAbstractType" >
597       <sequence>
598         <element ref="saml:Subject" />
599       </sequence>
600     </extension>
601   </complexContent>
602 </complexType>
```

603 **2.4.2.1. Element <Subject>**

604 The `<Subject>` element specifies the principal that is the subject of the statement element
605 specifies one or more subjects. It contains either or both of the following elements:
606 `<NameIdentifier>`
607 An identification of a subject by its name and security domain.
608 `<SubjectConfirmation>`
609 Information that allows the subject to be authenticated.
610 If the `<Subject>` element contains both a `<NameIdentifier>` and a
611 `<SubjectConfirmation>`, the issuer is asserting that if the relying party performs the specified
612 `<SubjectConfirmation>`, it can be confident that the entity presenting the assertion to the
613 relying party is the entity that the issuer associates with the `<NameIdentifier>`. If a `<Subject>`
614 element contains more than one subject specification, the issuer is asserting that the surrounding
615 statement is true for all of the subjects specified. For example, if both a `<NameIdentifier>` and a
616 `<SubjectConfirmation>` element are present, the issuer is asserting that the statement is true
617 of both subjects being identified. A `<Subject>` element SHOULD NOT identify more than one
618 principal.

619 The following schema fragment defines the `<Subject>` element and its **SubjectType** complex
620 type:

```
621 <element name="Subject" type="saml:SubjectType" />
622 <complexType name="SubjectType">
623   <choice>
624     <sequence>
625       <element ref="saml:NameIdentifier" />
626       <element ref="saml:SubjectConfirmation" minOccurs="0" />
627     </sequence>
628     <element ref="saml:SubjectConfirmation" />
629   </choice>
630 </complexType>
```

631 **2.4.2.2. Element <NameIdentifier>**

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

634 SecurityDomain **[Optional]**
635 The security domain governing the name of the subject.

636 Name **[Required]**
637 The name of the subject.

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

641 The following schema fragment defines the `<NameIdentifier>` element and its
642 **NameIdentifierType** complex type:

```
643 <element name="NameIdentifier" type="saml:NameIdentifierType" />
644 <complexType name="NameIdentifierType">
645   <attribute name="SecurityDomain" type="string" />
646   <attribute name="Name" type="string" use="required" />
647 </complexType>
```

648 **2.4.2.3. Elements <SubjectConfirmation>, <ConfirmationMethod>, and**
649 **<SubjectConfirmationData>**

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

652 `<ConfirmationMethod>` [One or more]

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

655 `<SubjectConfirmationData>` [Optional]

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

657 `<ds:KeyInfo>` [Optional]

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

660 The following schema fragment defines the `<SubjectConfirmation>` element and its
661 **SubjectConfirmationType** complex type, along with the `<SubjectConfirmationData>`
662 element and the `<ConfirmationMethod>` element:

```
663   <element name="SubjectConfirmation" type="saml:SubjectConfirmationType" />
664   <complexType name="SubjectConfirmationType">
665     <sequence>
666       <element ref="saml:ConfirmationMethod" maxOccurs="unbounded" />
667       <element ref="saml:SubjectConfirmationData" minOccurs="0" />
668       <element ref="ds:KeyInfo" minOccurs="0" />
669     </sequence>
670   </complexType>
671   <element name="SubjectConfirmationData" type="string" />
672   <element name="ConfirmationMethod" type="anyURI" />
```

673 **2.4.3. Element <AuthenticationStatement>**

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

678 `AuthenticationMethod` [Optional]

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

681 `AuthenticationInstant` [Optional]

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

684 `AuthenticationLocality` [Optional]

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

687 | `<Authorityentification>` Binding [Any Number]

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

689 The following schema fragment defines the `<AuthenticationStatement>` element and its
690 **AuthenticationStatementType** complex type:

```
691   <element name="AuthenticationStatement"
692     type="saml:AuthenticationStatementType" />
693   <complexType name="AuthenticationStatementType">
694     <complexContent>
695       <extension base="saml:SubjectStatementAbstractType" >
696         <sequence>
```

```

697             <element ref="saml:AuthenticationLocality" minOccurs="0" />
698             <element ref="saml:AuthorityBinding"
699                     minOccurs="0" maxOccurs="unbounded" />
700         </sequence>
701         <attribute name="AuthenticationMethod" type="anyURI" />
702         <attribute name="AuthenticationInstant" type="dateTime" />
703     </extension>
704     </complexContent>
705 </complexType>
```

706 2.4.3.1. Element <AuthenticationLocality>

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

709 IPAddress [Optional]

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

711 DNSAddress [Optional]

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

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

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

```

717     <element name="AuthenticationLocality"
718             type="saml:AuthenticationLocalityType" />
719     <complexType name="AuthenticationLocalityType" >
720         <attribute name="IPAddress" type="string" use="optional" />
721         <attribute name="DNSAddress" type="string" use="optional" />
722     </complexType>
```

723 2.4.3.2. Element <AuthorityBinding>

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

729 AuthorityKind [Required]

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

735 Location [Required]

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

739 Binding [Required]

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

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

```

744     <element name="AuthorityBinding" type="saml:AuthorityBindingType" />
745     <complexType name="AuthorityBindingType" >
```

```

746     <attribute name="AuthorityKind" type="saml:AuthorityKindType" use="required"/>
747
748     <attribute name="Location" type="anyURI" use="required"/>
749     <attribute name="Binding" type="anyURI" use="required"/>
750 </complexType>
751 <simpleType name="AuthorityKindType">
752     <restriction base="string">
753         <enumeration value="authentication"/>
754         <enumeration value="attribute"/>
755         <enumeration value="authorization"/>
756     </restriction>
757 </simpleType>

```

2.4.4. Element <AuthorizationDecisionStatement>

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

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

In general, the rules for equivalence and definition of a normal form, if any, are scheme dependent. When a scheme uses elements of the common syntax, it will also use the common syntax equivalence rules, namely that the scheme and hostname are case insensitive and a URL with an explicit ".port", where the port is the default for the scheme, is equivalent to one where the port is elided.

To avoid ambiguity resulting from variations in URI encoding SAML applications SHOULD employ the URI normalized form wherever possible as follows:

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

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

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

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

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

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

Resource [RequiredOptional]

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

Decision [RequiredOptional]

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

794 <Actions> [Required]
 795 The set of actions authorized to be performed on the specified resource.
 796 <Evidence> [Any Number]
 797 A set of assertions that the issuer relied on in making the decision.
 798 The following schema fragment defines the <AuthorizationDecisionStatement> element
 799 and its **AuthorizationDecisionStatementType** complex type:

```

800   <element name="AuthorizationDecisionStatement"
801     type="saml:AuthorizationDecisionStatementType" />
802     <complexType name="AuthorizationDecisionStatementType">
803       <complexContent>
804         <extension base="saml:SubjectStatementAbstractType">
805           <sequence>
806             <element ref="saml:Actions" />
807             <element ref="saml:Evidence" minOccurs="0"
808                   maxOccurs="unbounded" />
809           </sequence>
810           <attribute name="Resource" type="anyURI" use="required"
811         use="optional" />
812           <attribute name="Decision" type="saml:DecisionType"
813             use="required"use="optional" />
814         </extension>
815       </complexContent>
816     </complexType>

```

817 **2.4.4.1. Elements <Actions> and <Action>**

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

820 Namespace [Optional]
 821 A URI representing the namespace in which the names of specified actions are to be
 822 interpreted. If this element is absent, the namespace <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#rwedc-negation> specified in section
 823 7.2.2 is in effect.
 824

825 <Action> [One or more]
 826 An action sought to be performed on the specified resource.

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

```

829   <element name="Actions" type="saml:ActionsType" />
830   <complexType name="ActionsType">
831     <sequence>
832       <element ref="saml:Action" maxOccurs="unbounded" />
833     </sequence>
834     <attribute name="Namespace" type="anyURI" use="optional" />
835   </complexType>
836   <element name="Action" type="string" />

```

837 **2.4.4.2. Element <Evidence>**

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

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

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

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

847 2.4.5. Element <AttributeStatement>

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

851 <Attribute> [One or More]

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

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

```
855 <element name="AttributeStatement" type="saml:AttributeStatementType" />
856 <complexType name="AttributeStatementType">
857   <complexContent>
858     <extension base="saml:SubjectStatementAbstractType">
859       <sequence>
860         <element ref="saml:Attribute" maxOccurs="unbounded" />
861       </sequence>
862     </extension>
863   </complexContent>
864 </complexType>
```

865 2.4.5.1. Elements <AttributeDesignator> and <Attribute>

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

870 AttributeNamespace [Optional]

871 The namespace in which the AttributeName elements are interpreted.

872 AttributeName [Optional]

873 The name of the attribute.

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

```
876 <element name="AttributeDesignator" type="saml:AttributeDesignatorType" />
877 <complexType name="AttributeDesignatorType">
878   <attribute name="AttributeName" type="string" use="required"/>
879   <attribute name="AttributeNamespace" type="anyURI" use="required"/>
880 </complexType>
```

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

884 <AttributeValue> [Any Number]

885 The value of the attribute.

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

```
888 <element name="Attribute" type="saml:AttributeType" />
889 <complexType name="AttributeType">
890   <complexContent>
891     <extension base="saml:AttributeDesignatorType">
892       <sequence>
893         <element ref="saml:AttributeValue" maxOccurs="unbounded" />
```

```
894         </sequence>
895     </extension>
896   </complexContent>
897 </complexType>
```

898 **2.4.5.1.1 Element <AttributeValue>**

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

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

905 The following schema fragment defines the `<AttributeValue>` element:

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

3. SAML Protocol

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

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



914
915 Figure 2: SAML Request-Response Protocol

3.1. Schema Header and Namespace Declarations

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

```
919 <schema
920     targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-
921     sstc-schema-protocol-2726.xsd"
922     xmlns="http://www.w3.org/2001/XMLSchema"
923     xmlns:samlp="http://www.oasis-open.org/committees/security/docs/draft-sstc-
924     schema-protocol-2726.xsd"
925     xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-
926     schema-assertion-2726.xsd"
927     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
928     elementFormDefault="unqualified">
929     <import namespace="http://www.oasis-open.org/committees/security/docs/draft-
930     sstc-schema-assertion-2726.xsd"
931         schemaLocation="draft-sstc-schema-assertion-2726.xsd"/>
932     <import namespace="http://www.w3.org/2000/09/xmldsig#"
933         schemaLocation="xmldsig-core-schema.xsd"/>
934     <annotation>
935         <documentation>draft-sstc-schema-protocol-2726.xsd</documentation>
936     </annotation>
937     ...
938 </schema>
```

939

3.2. Requests

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

3.2.1. Complex Type RequestAbstractType

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

946 RequestID [Required]

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

950 MajorVersion [Required]
 951 The major version of this request. The identifier for the version of SAML defined in this
 952 specification is 1. Processing of this attribute is specified in Section 3.4.2.
 953 MinorVersion [Required]
 954 The minor version of this request. The identifier for the version of SAML defined in this
 955 specification is 0. Processing of this attribute is specified in Section 3.4.2.
 956 IssueInstant [Required]
 957 The time instant of issue of the request. The time value is encoded in UTC as described in
 958 section 1.2.1.
 959 <RespondWith> [Any Number]
 960 Each <RespondWith> element specifies a type of response that is acceptable to the
 961 requestor.
 962 <Signature> [Optional]
 963 An XML Signature that authenticates the assertion, see section 5.
 964 The following schema fragment defines the **RequestAbstractType** complex type:
 965

```
<complexType name="RequestAbstractType" abstract="true">
    <sequence>
        <element ref = "samlp:RespondWith"
            minOccurs="0" maxOccurs="unbounded" />
        <element ref = "ds:Signature" minOccurs="0" />
    </sequence>
    <attribute name="RequestID" type="saml:IDType" use="required" />
    <attribute name="MajorVersion" type="integer" use="required" />
    <attribute name="MinorVersion" type="integer" use="required" />
    <attribute name="IssueInstant" type="dateTime" use="required" />
</complexType>
```

976 **3.2.1.1. Element <RespondWith>**

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

979 The <RespondWith> element specifies the type(s) of response that is acceptable to the requestor.
 980 Multiple <RespondWith> elements MAY be specified to indicate that the requestor is capable of
 981 processing multiple requests.

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

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

989 [<RespondWith> element values are URIs. A requestor MAY use an XML schema identifier as a](#)
 990 [<RespondWith> element value to inform the responder that the specified SAML extension schema](#)
 991 [is supported. <RespondWith> values defined in this document are specified as URI fragment](#)
 992 [identifiers, the nominal base for these identifier values being the SAML protocol schema identifier](#)
 993 [URL.](#)

994 Acceptable values for the <RespondWith> element are:

995 #SingleStatement
 996 An assertion carrying exactly one statement element.

```

997 | #MultipleStatement  

998 |     An assertion carrying at least one statement element.  

999 | #AuthenticationStatement  

1000 |     An assertion carrying an Authentication statement.  

1001 | #AuthorizationDecisionStatement  

1002 |     An assertion carrying an Authorization Decision statement.  

1003 | #AttributeStatement  

1004 |     An assertion carrying an Attribute statement.  

1005 Schema URI  

1006     An assertion containing additional elements from the specified schema.  

1007 The following schema fragment defines the <RespondWith> element:  

1008 <element name="RespondWith" type="anyURI" />

```

1009 3.2.2. Element <Request>

1010 The <Request> element specifies a SAML request. It provides either a query or a request for a
1011 specific assertion identified by <AssertionIDReference> or <AssertionArtifact>. It has
1012 the complex type **RequestType**, which extends **RequestAbstractType** by adding a choice of one
1013 of the following elements:

```

1014 <Query>  

1015     An extension point that allows extension schemas to define new types of query.  

1016 <SubjectQuery>  

1017     An extension point that allows extension schemas to define new types of query that specify  

1018     a single SAML subject.  

1019 <AuthenticationQuery>  

1020     Makes a query for authentication information.  

1021 <AttributeQuery>  

1022     Makes a query for attribute information.  

1023 <AuthorizationDecisionQuery>  

1024     Makes a query for an authorization decision.  

1025 <AssertionIDReference> [One or more]  

1026     Requests an assertions by reference to its assertion identifier.  

1027 <AssertionArtifact> [One or more]  

1028     Requests an assertions by supplying an assertion artifact that represents it.

```

1029 The following schema fragment defines the <Request> element and its **RequestType** complex
1030 type:

```

1031 <element name="Request" type="samlp:RequestType" />  

1032 <complexType name="RequestType">  

1033     <complexContent>  

1034         <extension base="samlp:RequestAbstractType">  

1035             <choice>  

1036                 <element ref="samlp:Query" />  

1037                 <element ref="samlp:SubjectQuery" />  

1038                 <element ref="samlp:AuthenticationQuery" />  

1039                 <element ref="samlp:AttributeQuery" />  

1040                 <element ref="samlp:AuthorizationDecisionQuery" />  

1041                 <element ref="saml:AssertionIDReference" maxOccurs="unbounded" />  

1042                 <element ref="samlp:AssertionArtifact" maxOccurs="unbounded" />  

1043             </choice>  

1044         </extension>

```

```
1045     </complexContent>
1046 </complexType>
```

1047 3.2.3. Element <AssertionArtifact>

1048 The <AssertionArtifact> element is used to specify the assertion artifact that represents an
1049 assertion.

1050 The following schema fragment defines the <AssertionArtifact> element:

```
1051 <element name="AssertionArtifact" type="string" />
```

1052 3.3. Queries

1053 The following sections define the SAML constructs that contain query information.

1054 3.3.1. Element <Query>

1055 The <Query> element is an extension point that allows new SAML queries to be defined. Its
1056 **QueryAbstractType** is abstract; extension elements MUST use the xsi:type attribute to indicate
1057 the derived type. **QueryAbstractType** is the base type from which all SAML query elements are
1058 derived.

1059 The following schema fragment defines the <Query> element and its **QueryAbstractType**
1060 complex type:

```
1061 <element name="Query" type="samlp:QueryAbstractType" />
1062 <complexType name="QueryAbstractType" abstract="true" />
```

1063 3.3.2. Element <SubjectQuery>

1064 The <SubjectQuery> element is an extension point that allows new SAML queries that specify a
1065 single SAML subject. Its **SubjectQueryAbstractType** complex type is abstract; extension elements
1066 MUST use the xsi:type attribute to indicate the derived type. **SubjectQueryAbstractType** adds
1067 the <Subject> element.

1068 The following schema fragment defines the <SubjectQuery> element and its
1069 **SubjectQueryAbstractType** complex type:

```
1070 <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType" />
1071 <complexType name="SubjectQueryAbstractType" abstract="true" >
1072   <complexContent>
1073     <extension base="samlp:QueryAbstractType" >
1074       <sequence>
1075         <element ref="saml:Subject" />
1076       </sequence>
1077     </extension>
1078   </complexContent>
1079 </complexType>
```

1080 3.3.3. Element <AuthenticationQuery>

1081 The <AuthenticationQuery> element is used to make the query “What authentication
1082 assertions are available for this subject?” A successful response will be in the form of assertions
1083 containing authentication statements. This element is of type **AuthenticationQueryType**, which
1084 extends **SubjectQueryAbstractType** with the addition of the following element:

```
1085 <ConfirmationMethod> [Optional]
1086   A filter for possible responses. If it is present, the query made is “What authentication
1087   assertions do you have for this subject with the supplied confirmation method?”
```

1088 In response to an authentication query, a responder returns assertions with authentication
1089 statements as follows: The <Subject> element in the returned assertions MUST be identical to
1090 the <Subject> element of the query. If the <ConfirmationMethod> element is present in the
1091 query, at least one <ConfirmationMethod> element in the response MUST match. It is
1092 OPTIONAL for the complete set of all such matching assertions to be returned in the response.

1093 The following schema fragment defines the <AuthenticationQuery> type and its
1094 **AuthenticationQueryType** complex type:

```
1095 <element name="AuthenticationQuery" type="samlp:AuthenticationQueryType" />
1096 <complexType name="AuthenticationQueryType">
1097   <complexContent>
1098     <extension base="samlp:SubjectQueryAbstractType" >
1099       <sequence>
1100         <element ref="saml:ConfirmationMethod" minOccurs="0" />
1101       </sequence>
1102     </extension>
1103   </complexContent>
1104 </complexType>
```

1105 3.3.4. Element <AttributeQuery>

1106 The <AttributeQuery> element is used to make the query "Return the requested attributes for
1107 this subject." A successful response will be in the form of assertions containing attribute statements.
1108 This element is of type **AttributeQueryType**, which extends **SubjectQueryAbstractType** with the
1109 addition of the following element and attribute:

1110 Resource [Optional]
1111 The Resource attribute if present specifies that the attribute query is made in response to a
1112 specific authorization decision relating to the resource. The responder MAY use the
1113 resource attribute to establish the scope of the request.
1114 If the resource attribute is specified and the responder does not wish to support resource-
1115 specific attribute queries, or if the resource value provided is invalid or unrecognized, then it
1116 SHOULD respond with a SAML status of "Error.Receiver.ResourceNotRecognized".

1117 <AttributeDesignator> [Any Number] (see Section 2.4.5.1)
1118 Each <AttributeDesignator> element specifies an attribute whose value is to be
1119 returned. If no attributes are specified, the list of desired attributes is implicit and
1120 application-specific.

1121 The following schema fragment defines the <AttributeQuery> element and its
1122 **AttributeQueryType** complex type:

```
1123 <element name="AttributeQuery" type="samlp:AttributeQueryType" />
1124 <complexType name="AttributeQueryType">
1125   <complexContent>
1126     <extension base="samlp:SubjectQueryAbstractType" >
1127       <sequence>
1128         <element ref="saml:AttributeDesignator"
1129           minOccurs="0" maxOccurs="unbounded" />
1130       </sequence>
1131       <attribute name="Resource" type="anyURI" use="optional" />
1132     </extension>
1133   </complexContent>
1134 </complexType>
```

1135 3.3.5. Element <AuthorizationDecisionQuery>

1136 The <AuthorizationDecisionQuery> element is used to make the query "Should these
1137 actions on this resource be allowed for this subject, given this evidence?" A successful response
1138 will be in the form of assertions containing authorization decision statements. This element is of

1139 type **AuthorizationDecisionQueryType**, which extends **SubjectQueryAbstractType** with the
1140 addition of the following elements and attribute:

1141 Resource [Required]
1142 A URI indicating the resource for which authorization is requested.

1143 <Actions> [Required]
1144 The actions for which authorization is requested.

1145 <Evidence> [Any Number]
1146 An assertion that the responder MAY rely on in making its response.

1147 The following schema fragment defines the `<AuthorizationDecisionQuery>` element and its
1148 **AuthorizationDecisionQueryType** complex type:

```
1149 <element name="AuthorizationDecisionQuery"  
1150 type="samlp:AuthorizationDecisionQueryType"/>  
1151 <complexType name="AuthorizationDecisionQueryType">  
1152   <complexContent>  
1153     <extension base="samlp:SubjectQueryAbstractType">  
1154       <sequence>  
1155         <element ref="saml:Actions" />  
1156         <element ref="saml:Evidence"  
1157           minOccurs="0" maxOccurs="unbounded" />  
1158       </sequence>  
1159       <attribute name="Resource" type="anyURI" use="required" />  
1160     </extension>  
1161   </complexContent>  
1162 </complexType>
```

1163 3.4. Responses

1164 The following sections define the SAML constructs that contain response information.

1165 3.4.1. Complex Type ResponseAbstractType

1166 All SAML responses are of types that are derived from the abstract **ResponseAbstractType**
1167 complex type. This type defines common attributes and elements that are associated with all SAML
1168 responses:

1169 ResponseID [Required]
1170 An identifier for the response. It is of type **IDType**, and MUST follow the requirements
1171 specified by that type for identifier uniqueness.

1172 InResponseTo [Required]
1173 A reference to the identifier of the request to which the response corresponds. The value of
1174 this attribute MUST match the value of the corresponding RequestID attribute.

1175 MajorVersion [Required]
1176 The major version of this response. The identifier for the version of SAML defined in this
1177 specification is 1. Processing of this attribute is specified in Section 3.4.4.

1178 MinorVersion [Required]
1179 The minor version of this response. The identifier for the version of SAML defined in this
1180 specification is 0. Processing of this attribute is specified in Section 3.4.4.

1181 IssueInstant [Optional]
1182 The time instant of issue of the request. The time value is encoded in UTC as described in
1183 section 1.2.1.

1184 <Signature> [OptionalAny Number]
1185 An XML Signature that authenticates the assertion, see section 5.

1186 The following schema fragment defines the **ResponseAbstractType** complex type:

```
1187 <complexType name="ResponseAbstractType" abstract="true">
1188     <sequence>
1189         <element ref = "ds:Signature" minOccurs="0" />
1190     </sequence>
1191     <attribute name="ResponseID" type="saml:IDType" use="required"/>
1192     <attribute name="InResponseTo" type="saml:IDReferenceType"
1193         use="required"/>
1194     <attribute name="MajorVersion" type="integer" use="required"/>
1195     <attribute name="MinorVersion" type="integer" use="required"/>
1196     <attribute name="IssueInstant" type="dateTime" use="required"/>
1197 </complexType>
```

1198 3.4.2. Element <Response>

1199 The <Response> element specifies the status of the corresponding SAML request and a list of
1200 zero or more assertions that answer the request. It has the complex type **ResponseType**, which
1201 extends **ResponseAbstractType** by adding the following elements (in an unbounded mixture):

```
1202 <Status> [Required] (see Section 3.4.3)
1203     A code representing the status of the corresponding request.

1204 <Assertion> [Any Number] (see Section 2.3.3)
1205     Specifies an assertion by value.
```

1206 The following schema fragment defines the <Response> element and its **ResponseType** complex
1207 type:

```
1208 <element name="Response" type="samlp:ResponseType" />
1209 <complexType name="ResponseType">
1210     <complexContent>
1211         <extension base="samlp:ResponseAbstractType">
1212             <sequence>
1213                 <element ref="samlp:Status" />
1214                 <element ref="saml:Assertion"
1215                     minOccurs="0" maxOccurs="unbounded" />
1216             </sequence>
1217         </extension>
1218     </complexContent>
1219 </complexType>
```

1220 3.4.3. Element <Status>

1221 The <Status> element :

```
1222 <StatusCode> [Required]
1223     A code representing the status of the corresponding request.

1224 <StatusMessage> [Any Number]
1225     A message which MAY be returned to an operator.

1226 <StatusDetail> [Optional]
1227     Specifies additional information concerning an error condition.
```

1228 The following schema fragment defines the <Status> element and its **StatusType** complex type:

```
1229 <element name="Status" type="samlp:StatusType" />
1230 <complexType name="StatusType">
1231     <sequence>
1232         <element ref="samlp:StatusCode" />
1233         <element ref="samlp:StatusMessage"
1234             minOccurs="0" maxOccurs="unbounded" />
1235         <element ref="samlp:StatusDetail" minOccurs="0" />
```

```
1236      </sequence>
1237  </complexType>
```

1238 3.4.3.1. Element <StatusCode>

1239 The <StatusCode> element specifies a code representing the status of the corresponding request
1240 and an option sub code providing more specific information concerning a particular error status:

1241 Value [Required]

The status code value as defined below.

1243 <SubStatusCode> [Optional]

1244 An optional subordinate status code value that provides more specific information on an
1245 error condition.

1246 The following **StatusCode** values are defined:

1247 Success

1248 The request succeeded.

1249 VersionMismatch

1250 The receiver could not process the request because the version was incorrect.

1251 Receiver

1252 The request could not be performed due to an error at the receiving end.

1253 Sender

1254 The request could not be performed due to an error in the sender or in the request

1255 The following schema fragment defines the <StatusCode> element and its **StatusCodeType**
1256 complex type and the **StatusCodeEnumType** simple type:

```
1257 <element name="StatusCode" type="samlp:StatusCodeType" />
1258 <complexType name="StatusCodeType">
1259   <sequence>
1260     <element ref="samlp:SubStatusCode" minOccurs="0" />
1261   </sequence>
1262   <attribute name="Value" type="samlp:StatusCodeEnumType" use="required" />
1263 </complexType>
1264 <simpleType name="StatusCodeEnumType">
1265   <restriction base="QName">
1266     <enumeration value="samlp:Success" />
1267     <enumeration value="samlp:VersionMismatch" />
1268     <enumeration value="samlp:Receiver" />
1269     <enumeration value="samlp:Sender" />
1270   </restriction>
1271 </simpleType>
```

1272 3.4.3.2. Element <SubStatusCode>

1273 The <SubStatusCode> element specifies an additional code representing the status of the
1274 corresponding request:

1275 Value [Required]

The status code value as defined below.

1277 <SubStatusCode> [Optional]

1278 An optional subordinate status code value that provides an additional level of specific
1279 information on an error condition.

1280 The following **SubStatusCode** values are defined, additional codes MAY be defined in future
1281 versions of the SAML specification:

```
1282 RequestVersionTooHigh
1283     The protocol version specified in the request is a major upgrade from the highest protocol
1284     version supported by the responder.
1285 RequestVersionTooLow
1286     The responder cannot respond to the particular request using the SAML version specified
1287     in the request because it is too low.
1288 RequestVersionDeprecated
1289     The responder does not respond to any requests with the protocol version specified in the
1290     request.
1291 TooManyResponses
1292     The response would contain more elements than the responder will return.
```

1293 The following schema fragment defines the `<SubStatusCode>` element and its
1294 **SubStatusCodeType** complex type:

```
1295 <element name="SubStatusCode" type="samlp:SubStatusCodeType" />
1296 <complexType name="SubStatusCodeType">
1297     <sequence>
1298         <element ref="samlp:SubStatusCode" minOccurs="0" />
1299     </sequence>
1300     <attribute name="Value" type="QName" use="required" />
1301 </complexType>
```

1302 **3.4.3.3. Element <StatusMessage>**

1303 The `<StatusMessage>` element specifies a message that MAY be returned to an operator:

1304 The following schema fragment defines the `<StatusMessage>` element and its
1305 **StatusMessageType** complex type:

```
1306 <element name="StatusMessage" type="string" />
```

1307 **3.4.3.4. Element <StatusDetail>**

1308 The `<StatusDetail>` element MAY be used to specify additional information concerning an error
1309 condition.

1310 The following schema fragment defines the `<StatusDetail>` element and its **StatusDetailType**
1311 complex type:

```
1312 <element name="StatusDetail" type="samlp:StatusDetailType" />
1313 <complexType name="StatusDetailType">
1314     <sequence>
1315         <any namespace="##any"
1316             processContents="lax" minOccurs="0" maxOccurs="unbounded" />
1317     </sequence>
1318 </complexType>
```

1319 **3.4.4. Responses to <AuthenticationQuery> and <AttributeQuery>**

1320 Responses to Authentication and Attribute queries are constructed by matching against the
1321 `<saml:Subject>` element found within the `<AuthenticationQuery>` or `<AttributeQuery>`
1322 elements. In response to these queries, every assertion returned by a SAML responder MUST
1323 contain at least one statement whose `<saml:Subject>` element **strongly matches** the
1324 `<saml:Subject>` element found in the query.

1325 A `<saml:Subject>` element S1 strongly matches S2 if and only if:

- 1 If S2 includes a `<saml:NameIdentifier>` element, then S1 must include an identical
`<saml:NameIdentifier>` element.

- 1328 2 If S2 includes a <saml:SubjectConfirmation> element, then S1 must include an
1329 identical <saml:SubjectConfirmation> element.

4. SAML Versioning

- 1330 SAML version information appears in the following elements:
- 1331 ?? <Assertion>
- 1332 ?? <Request>
- 1333 ?? <Response>
- 1334 The version numbering of the SAML assertion is independent of the version number of the SAML request-response protocol. The version information for each consists of a major version number and a minor version number, both of which are integers. In accordance with industry practice a version number SHOULD be presented to the user in the form *Major.Minor*. This document defines SAML Assertions 1.0 and SAML Protocol 1.0.
- 1335 The version number *Major_B.Minor_B* is higher than the version number *Major_A.Minor_A* if and only if:
- 1336 $Major_B > Major_A \text{ ? } ((Major_B = Major_A) \text{ ? } Minor_B > Minor_A)$
- 1337 Each revision of SAML SHALL assign version numbers to assertions, requests, and responses that are the same as or higher than the corresponding version number in the SAML version that immediately preceded it.
- 1338 New versions of SAML SHALL assign new version numbers as follows:
- 1339 ?? **Documentation change:** ($Major_B = Major_A$) ? ($Minor_B > Minor_A$)
1340 If the major and minor version numbers are unchanged, the new version *B* only introduces changes to the documentation that raise no compatibility issues with an implementation of version *A*.
- 1341 ?? **Minor upgrade:** ($Major_B = Major_A$) ? ($Minor_B > Minor_A$)
1342 If the major version number of versions *A* and *B* are the same and the minor version number of *B* is higher than that of *A*, the new SAML version MAY introduce changes to the SAML schema and semantics but any changes that are introduced in *B* SHALL be compatible with version *A*.
- 1343 ?? **Major upgrade:** $Major_B > Major_A$
1344 If the major version of *B* number is higher than the major version of *A*, Version *B* MAY introduce changes to the SAML schema and semantics that are incompatible with *A*.

4.1. Assertion Version

- 1345 A SAML application MUST NOT issue any assertion whose version number is not supported.
- 1346 A SAML application MUST reject any assertion whose major version number is not supported.
- 1347 A SAML application MAY reject any assertion whose version number is higher than the highest supported version.

4.2. Request Version

- 1348 A SAML application SHOULD issue requests that specify the highest SAML version supported by both the sender and recipient.
- 1349 If the SAML application does not know the capabilities of the recipient it should assume that it supports the highest SAML version supported by the sender.

1368 **4.3. Response Version**

- 1369 A SAML application MUST NOT issue responses that specify a higher SAML version number than
1370 the corresponding request.
- 1371 A SAML application MUST NOT issue a response that has a major version number that is lower
1372 than the major version number of the corresponding request except to report the error
1373 RequestVersionTooHigh.
- 1374 Incompatible protocol versions MAY cause the following errors to be reported:
- 1375 RequestVersionTooHigh
1376 The protocol version specified in the request is a major upgrade from the highest protocol
1377 version supported by the responder.
- 1378 RequestVersionTooLow
1379 The responder cannot respond to the particular request using the SAML version specified
1380 in the request because it is too low.
- 1381 RequestVersionDeprecated
1382 The responder does not respond to any requests with the protocol version specified in the
1383 request.

1384 5. SAML & XML-Signature Syntax and 1385 Processing

1386 SAML Assertions, Request and Response messages may be signed, with the following benefits:

1387 ?? An Assertion signed by the issuer (AP). This supports :

1388 (1) Message integrity

1389 (2) Authentication of the issuer to a relying party

1390 (3) If the signature is based on the issuer's public-private key pair, then it also
1391 provides for non-repudiation of origin.

1392 ?? A SAML request or a SAML response message signed by the message originator. This
1393 supports :

1394 (1) Message integrity

1395 (2) Authentication of message origin to a destination

1396 (3) If the signature is based on the originator's public-private key pair, then it also
1397 provides for non-repudiation of origin.

1398 Note :

1399 ?? SAML documents may be the subject of signatures from different packaging contexts.

1400 [XMLSig] provides a framework for signing in XML and is the framework of choice.

1401 However, signing may also take place in the context of S/MIME or Java objects that
1402 contain SAML documents. One goal is to ensure compatibility with this type of "foreign"
1403 digital signing.

1404 ?? It is useful to characterize situations when a digital signature is NOT required in SAML.

1405 Assertions:

1406 The asserting party has provided the assertion to the relying party, authenticated by means
1407 other than digital signature and the channel is secure. In other words, the RP has obtained the
1408 assertion from the AP directly (no intermediaries) through a secure channel and the AP has
1409 authenticated to the RP.

1410 Request/Response messages:

1411 The originator has authenticated to the destination and the destination has obtained the
1412 assertion directly from the originator (no intermediaries) through secure channel(s).

1413 Many different techniques are available for "direct" authentication and secure channel between
1414 two parties. The list includes SSL, HMAC, password-based login etc. Also the security
1415 requirement depends on the communicating applications and the nature of the assertion
1416 transported.

1417 All other contexts require the use of digital signature for assertions and request and response
1418 messages. Specifically:

1419 (1) An assertion obtained by a relying party from an entity other than the asserting party MUST
1420 be signed by the issuer.

1421 (2) A SAML message arriving at a destination from an entity other than the originating site
1422 MUST be signed by the origin site.

1423 5.1. Signing Assertions

1424 All SAML assertions MAY be signed using the XML Signature. This is reflected in the assertion
1425 schema – Section 2.3.

1426 **5.2. Request/Response Signing**

1427 All SAML requests and responses MAY be signed using the XML Signature. This is reflected in the
1428 schema – Section 3.2 & 3.4.

1429 **5.3. Signature Inheritance**

1430 **5.3.1. Rationale**

1431 SAML assertions may be embedded within request or response messages or other XML
1432 messages, which may be signed. Request or response messages may themselves be contained
1433 within other messages that are based on other XML messaging frameworks (e.g., SOAP) and the
1434 composite object may be the subject of a signature. Another possibility is that SAML assertions or
1435 request/response messages are embedded within a non-XML messaging object (e.g., MIME
1436 package) and signed.

1437 In such a case, the SAML sub-message (Assertion, request, response) may be viewed as inheriting
1438 a signature from the "super-signature" over the enclosing object, provided certain constraints are
1439 met.

1440 (1) An assertion may be viewed as inheriting a signature from a super signature, if the super
1441 signature applies all the elements within the assertion.

1442 A SAML request or response may be viewed as inheriting a signature from a super signature, if the
1443 super signature applies to all of the elements within the response.

1444 **5.3.2. Rules for SAML Signature Inheritance**

1445 Signature inheritance occurs when SAML message (assertion/request/response) is not signed but
1446 is enclosed within signed SAML such that the signature applies to all of the elements within the
1447 message. In such a case, the SAML message is said to inherit the signature and may be
1448 considered equivalent to the case where it is explicitly signed. The SAML message inherits the
1449 "closest enclosing signature".

1450 But if SAML messages need to be passed around by themselves, or embedded in other messages,
1451 they would need to be signed as per section 5.1

1452 **5.4. XML Signature Profile**

1453 The XML Signature [XMLSig] specification calls out a general XML syntax for signing data with
1454 many flexibilities and choices. This section details the constraints on these facilities so that SAML
1455 processors do not have to deal with the full generality of XML Signature processing.

1456 **5.4.1. Signing formats**

1457 XML Signature has three ways of representing signature in a document viz: enveloping, enveloped
1458 and detached.

1459 SAML assertions and protocols MUST use the enveloped signatures for signing assertions and
1460 protocols. SAML processors should support use of RSA signing and verification for public key
1461 operations.

1462 **5.4.2. CanonicalizationMethod**

1463 XML Signature REQUIRES the Canonical XML (omits comments)
1464 (<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>). SAML implementations SHOULD use
1465 Canonical XML with no comments.

1466 **5.4.3. Transforms**

1467 [XMLSig] REQUIRES the enveloped signature transform
1468 <http://www.w3.org/2000/09/xmldsig#enveloped-signature>

1469 **5.4.4. KeyInfo**

1470 SAML does not restrict or impose any restrictions in this area. Therefore following [XMLSig]
1471 keyInfo may be absent.

1472 **5.4.5. Binding between statements in a multi-statement assertion**

1473 Use of signing does not affect semantics of statements within assertions in any way, as stated in
1474 this document Sections 1 through 4.

6. SAML Extensions

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

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

6.1. Assertion Schema Extension

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

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

```
?? <Assertion>
?? <Condition>
?? <Statement>
?? <SubjectStatement>
?? <AdviceElement>
```

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

```
?? <AuthenticationStatement>
?? <AuthorizationDecisionStatement>
?? <AttributeStatement>
?? <AudienceRestrictionCondition>
```

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

```
?? <AttributeValue>
?? <Advice>
```

6.2. Protocol Schema Extension

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

```
?? <Query>
?? <SubjectQuery>
```

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

```
?? <Request>
```

```
1511    ?? <AuthenticationQuery>
1512    ?? <AuthorizationDecisionQuery>
1513    ?? <AttributeQuery>
1514    ?? <Response>
```

1515 6.3. Use of Type Derivation and Substitution Groups

1516 W3C XML Schema [Schema1] provides two principal mechanisms for specifying an element of an
1517 extended type: type derivation and substitution groups.

1518 For example, a `<Statement>` element can be assigned the type **NewStatementType** by means of
1519 the `xsi:type` attribute. For such an element to be schema-valid, **NewStatementType** needs to be
1520 derived from **StatementType**. The following example of a SAML assertion assumes that the
1521 extension schema (represented by the `new:` prefix) has defined this new type:

```
1522 <saml:Assertion ...>
1523   <saml:Statement xsi:type="new:NewStatementType">
1524     ...
1525   </saml:Statement>
1526 </saml:Assertion>
```

1527 Alternatively, the extension schema can define a `<NewStatement>` element that is a member of a
1528 substitution group that has `<Statement>` as a head element. For the substituted element to be
1529 schema-valid, it needs to have a type that matches or is derived from the head element's type. The
1530 following is an example of an extension schema fragment that defines this new element:

```
1531 <xsd:element "NewStatement" type="new:NewStatementType"
1532   substitutionGroup="saml:Statement" />
```

1533 The substitution group declaration allows the `<NewStatement>` element to be used anywhere the
1534 SAML `<Statement>` element can be used. The following is an example of a SAML assertion that
1535 uses the extension element:

```
1536 <saml:Assertion ...>
1537   <new:NewStatement>
1538     ...
1539   </new:NewStatement>
1540 </saml:Assertion>
```

1541 The choice of extension method has no effect on the semantics of the XML document but does
1542 have implications for interoperability.

1543 The advantages of type derivation are as follows:

- 1544 ?? A document can be more fully interpreted by a parser that does not have access to the
1545 extension schema because a “native” SAML element is available.
- 1546 ?? At the time of writing, some W3C XML Schema validators do not support substitution
1547 groups, whereas the `xsi:type` attribute is widely supported.

1548 The advantage of substitution groups is that a document can be explained without the need to
1549 explain the functioning of the `xsi:type` attribute.

1550 7. SAML-Defined Identifiers

1551 The following sections define URI-based identifiers for common authentication protocols and
1552 actions.
1553 Where possible an existing URN is used to specify a protocol. In the case of IETF protocols the
1554 URN of the most current RFC that specifies the protocol is used. URLs created specifically for
1555 SAML have the initial stem:
1556 <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-2726>

1557 7.1. Confirmation Method Identifiers

1558 The following identifiers MAY be used in the <ConfirmationMethod> element (see Section
1559 2.4.2.3) to refer to common authentication protocols.

1560 7.1.1. SAML Artifact:

1561 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#artifact>
1562 <SubjectConfirmationData>: Base64 (*Artifact*)
1563 The subject of the assertion is the party that can present the SAML Artifact value specified in
1564 <SubjectConfirmationData>.

1565 7.1.2. SAML Artifact (SHA-1):

1566 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#artifact-sha1>
1567 <SubjectConfirmationData>: Base64 (SHA1(*Artifact*))
1568 The subject of the assertion is the party that can present a SAML Artifact such that the SHA1 digest
1569 of the specified artifact matches the value specified in <SubjectConfirmationData> .

1570 7.1.3. Holder of Key:

1571 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#Holder-Of-Key>
1572 <ds:KeyInfo>: Any cryptographic key
1573 The subject of the assertion is the party that can demonstrate that it is the holder of the private
1574 component of the key specified in <ds:KeyInfo>.

1575 7.1.4. Sender Vouches:

1576 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#sender-vouches>
1577 Indicates that no other information is available about the context of use of the assertion. The
1578 Relying party SHOULD utilize other means to determine if it should process the assertion further.

1579 7.1.5. Password (Pass-Through):

1580 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#password>
1581 <SubjectConfirmationData>: Base64 (*Password*)

1582 The subject of the assertion is the party that can present the password value specified in
1583 <SubjectConfirmationData>.

1584 The username of the subject is specified by means of the <NameIdentifier> element.

1585 **7.1.6. Password (One-Way-Function SHA-1):**

1586 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#password-sha1>

1587 <SubjectConfirmationData>: *Base64 (SHA1 (Password))*

1588 The subject of the assertion is the party that can present the password such that the SHA1 digest of
1589 the specified password matches the value specified in <SubjectConfirmationData>.

1590 The username of the subject is specified by means of the <NameIdentifier> element.

1591 **7.1.7. Kerberos**

1592 **URI:** <urn:ietf:rfc:1510>

1593 <SubjectConfirmationData>: A Kerberos Ticket

1594 The subject is authenticated by means of the Kerberos protocol [RFC 1510], an instantiation of the
1595 Needham-Schroeder symmetric key authentication mechanism [Needham78].

1596 **7.1.8. SSL/TLS Certificate Based Client Authentication:**

1597 **URI:** <urn:ietf:rfc:2246>

1598 <ds:KeyInfo>: Any cryptographic key

1599 **7.1.9. Object Authenticator (SHA-1):**

1600 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#object-sha1>

1601 <SubjectConfirmationData>: *Base64 (SHA1 (Object))*

1602 This authenticator element is the result of computing a digest, using the SHA-1 hash algorithm. It is
1603 used when the subject can be represented as a binary string, for example when it is an XML
1604 document or the disk image of executable code. Any preprocessing of the subject prior to
1605 computation of the digest is out of scope. The name of the subject should be conveyed in an
1606 accompanying Nameldentifier element.

1607 **7.1.10. PKCS#7**

1608 **URI:** <urn:ietf:rfc:2315>

1609 <SubjectConfirmationData>: *Base64 (PKCS#7 (Object))*

1610 This authenticator element is signed data in PKCS#7 format [PKCS#7]. The posited identity of the
1611 signer must be conveyed in an accompanying Nameldentifier element. This subject type may be
1612 included in the subject field of an authentication query, in which case the corresponding response
1613 indicates whether the posited signer is, indeed, the signer. It may be included in an attribute query,
1614 in which case, the requested attribute values for the subject authenticated by the signed data are
1615 returned. It may be included in an authorization query, in which case, the access request
1616 represented by the signed data shall be identified by the accompanying object element, and the

1617 corresponding authorization decision assertion indicates whether the signer is authorized for the
1618 access request represented by the object element.

1619 **7.1.11. Cryptographic Message Syntax**

1620 **URI:** urn:ietf:rfc:2630

1621 <SubjectConfirmationData>: Base64 (CMS (Object))

1622 This authenticator element is signed data in CMS format [CMS]. See also 7.1.10

1623 **7.1.12. XML Digital Signature**

1624 **URI:** urn:ietf:rfc:3075

1625 <SubjectConfirmationData>: Base64 (XML-SIG (Object))

1626 <ds:KeyInfo>: A cryptographic signing key

1627 This authenticator element is signed data in XML Signature format. See also 7.1.10

1628 **7.2. Action Namespace Identifiers**

1629 The following identifiers MAY be used in the ActionNamespace attribute (see Section 2.4.4.1) to
1630 refer to common sets of actions to perform on resources.

1631 **7.2.1. Read/Write/Execute/Delete/Control:**

1632 **URI:** http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#rwedc

1633 Defined actions:

1634 Read Write Execute Delete Control

1635 These actions are interpreted in the normal manner, i.e.

1636 Read

1637 The subject may read the resource

1638 Write

1639 The subject may modify the resource

1640 Execute

1641 The subject may execute the resource

1642 Delete

1643 The subject may delete the resource

1644 Control

1645 The subject may specify the access control policy for the resource

1646 **7.2.2. Read/Write/Execute/Delete/Control with Negation:**

1647 **URI:** http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#rwedc-negation

1648 Defined actions:

1649 Read Write Execute Delete Control ~Read ~Write ~Execute ~Delete ~Control

1650 The actions specified in section 7.2.1 are interpreted in the same manner described there. Actions
1651 prefixed with a tilde ~ are negated permissions and are used to affirmatively specify that the stated
1652 permission is denied. Thus a subject described as being authorized to perform the action ~Read is
1653 affirmatively denied read permission.

1654 An application MUST NOT authorize both an action and its negated form.

1655 **7.2.3. Get/Head/Put/Post:**

1656 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#ghpp>

1657 Defined actions:

1658 GET HEAD PUT POST

1659 These actions bind to the corresponding HTTP operations. For example a subject authorized to
1660 perform the GET action on a resource is authorized to retrieve it.

1661 The GET and HEAD actions loosely correspond to the conventional read permission and the PUT
1662 and POST actions to the write permission. The correspondence is not exact however since a HTTP
1663 GET operation may cause data to be modified and a POST operation may cause modification to a
1664 resource other than the one specified in the request. For this reason a separate Action URI
1665 specifier is provided.

1666 **7.2.4. UNIX File Permissions:**

1667 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#unix>

1668 The defined actions are the set of UNIX file access permissions expressed in the numeric (octal)
1669 notation.

1670 The action string is a four digit numeric code:

1671 *extended user group world*

1672 Where the *extended* access permission has the value

1673 +2 if sgid is set

1674 +4 if suid is set

1675 The *user group* and *world* access permissions have the value

1676 +1 if execute permission is granted

1677 +2 if write permission is granted

1678 +4 if read permission is granted

1679 For example 0754 denotes the UNIX file access permission: user read, write and execute, group
1680 read and execute and world read.

1681 8. SAML Schema Listings

1682 The following sections contain complete listings of the assertion and protocol schemas for SAML.

1683 8.1. Assertion Schema

1684 Following is a complete listing of the SAML assertion schema [SAML-XSD].

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-Baker
(VeriSign Inc.) -->
<schema
    targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-
sstc-schema-assertion-2726.xsd"
    xmlns="http://www.w3.org/2001/XMLSchema" xmlns:saml="http://www.oasis-
open.org/committees/security/docs/draft-sstc-schema-assertion-2726.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    elementFormDefault="unqualified">
    <import namespace="http://www.w3.org/2000/09/xmldsig#"
        schemaLocation="xmldsig-core-schema.xsd"/>
    <annotation>
        <documentation>draft-sstc-schema-assertion-2726.xsd</documentation>
    </annotation>
    <simpleType name="IDType">
        <restriction base="string"/>
    </simpleType>
    <simpleType name="IDReferenceType">
        <restriction base="string"/>
    </simpleType>
    <simpleType name="DecisionType">
        <restriction base="string">
            <enumeration value="Permit"/>
            <enumeration value="Deny"/>
            <enumeration value="Indeterminate"/>
        </restriction>
    </simpleType>
    <element name="AssertionSpecifier" type="saml:AssertionSpecifierType" />
    <complexType name="AssertionSpecifierType">
        <choice>
            <element ref="saml:AssertionIDReference" />
            <element ref="saml:Assertion" />
        </choice>
    </complexType>
    <element name="AssertionIDReference" type="saml:IDReferenceType" />
    <element name="Assertion" type="saml:AssertionType" />
    <complexType name="AssertionType">
        <sequence>
            <element ref="saml:Conditions" minOccurs="0" />
            <element ref="saml:Advice" minOccurs="0" />
            <choice maxOccurs="unbounded">
                <element ref="saml:Statement" />
                <element ref="saml:SubjectStatement" />
                <element ref="saml:AuthenticationStatement" />
                <element ref="saml:AuthorizationDecisionStatement" />
                <element ref="saml:AttributeStatement" />
            </choice>
            <element ref = "ds:Signature" minOccurs="0" />
        </sequence>
        <attribute name="MajorVersion" type="integer" use="required" />
        <attribute name="MinorVersion" type="integer" use="required" />
        <attribute name="AssertionID" type="saml:IDType" use="required" />
```

```

1738     <attribute name="Issuer" type="string" use="required"/>
1739     <attribute name="IssueInstant" type="dateTime" use="required"/>
1740 </complexType>
1741 <element name="Conditions" type="saml:ConditionsType" />
1742 <complexType name="ConditionsType">
1743     <choice minOccurs="0" maxOccurs="unbounded">
1744         <element ref="saml:Condition" />
1745         <element ref="saml:AudienceRestrictionCondition" />
1746     </choice>
1747     <attribute name="NotBefore" type="dateTime" use="optional"/>
1748     <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>
1749 </complexType>
1750 <element name="Condition" type="saml:ConditionAbstractType" />
1751 <complexType name="ConditionAbstractType" abstract="true" />
1752 <element name="AudienceRestrictionCondition" type="saml:AudienceRestrictionConditionType" />
1753 <complexType name="AudienceRestrictionConditionType" >
1754     <complexContent>
1755         <extension base="saml:ConditionAbstractType" >
1756             <sequence>
1757                 <element ref="saml:Audience" maxOccurs="unbounded" />
1758             </sequence>
1759         </extension>
1760     </complexContent>
1761 </complexType>
1762 <element name="Audience" type="anyURI" />
1763 <element name="TargetRestrictionCondition" type="saml:TargetRestrictionConditionType" />
1764 <complexType name="TargetRestrictionConditionType" >
1765     <complexContent>
1766         <extension base="saml:ConditionAbstractType" >
1767             <sequence>
1768                 <element ref="saml:Target" minOccurs="1" maxOccurs="unbounded" />
1769             </sequence>
1770         </extension>
1771     </complexContent>
1772 </complexType>
1773 <element name="Target" type="anyURI" />
1774 <element name="Advice" type="saml:AdviceType" />
1775 <complexType name="AdviceType" >
1776     <sequence>
1777         <choice minOccurs="0" maxOccurs="unbounded" >
1778             <element ref="saml:AssertionSpecifier" />
1779             <element ref="saml:AdviceElement" />
1780             <any namespace="##other" processContents="lax" />
1781         </choice>
1782     </sequence>
1783 </complexType>
1784 <complexType name="AdviceElement" type="saml:AdviceAbstractType" />
1785 <complexType name="AdviceAbstractType" />
1786 <element name="Statement" type="saml:StatementAbstractType" />
1787 <complexType name="StatementAbstractType" abstract="true" />
1788 <element name="SubjectStatement" type="saml:SubjectStatementAbstractType" />
1789 <complexType name="SubjectStatementAbstractType" abstract="true" >
1790     <complexContent>
1791         <extension base="saml:StatementAbstractType" >
1792             <sequence>
1793                 <element ref="saml:Subject" />
1794             </sequence>
1795         </extension>
1796     </complexContent>
1797 </complexType>
1798 <complexType name="StatementAbstractType" >
1799     <complexContent>
1800         <extension base="saml:StatementAbstractType" >

```

```

1801 <element name="Subject" type="saml:SubjectType" />
1802 <complexType name="SubjectType">
1803   <choice>
1804     <sequence>
1805       <element ref="saml:NameIdentifier" />
1806       <element ref="saml:SubjectConfirmation" minOccurs="0" />
1807     </sequence>
1808     <element ref="saml:SubjectConfirmation" />
1809   </choice>
1810 </complexType>
1811 <element name="NameIdentifier" type="saml:NameIdentifierType" />
1812 <complexType name="NameIdentifierType">
1813   <attribute name="SecurityDomain" type="string" />
1814   <attribute name="Name" type="string" use="required" />
1815 </complexType>
1816 <element name="SubjectConfirmation" type="saml:SubjectConfirmationType" />
1817 <complexType name="SubjectConfirmationType">
1818   <sequence>
1819     <element ref="saml:ConfirmationMethod" maxOccurs="unbounded" />
1820     <element ref="saml:SubjectConfirmationData" minOccurs="0" />
1821     <element ref="ds:KeyInfo" minOccurs="0" />
1822   </sequence>
1823 </complexType>
1824 <element name="SubjectConfirmationData" type="string" />
1825 <element name="ConfirmationMethod" type="anyURI" />
1826 <element name="AuthenticationStatement"
1827   type="saml:AuthenticationStatementType" />
1828 <complexType name="AuthenticationStatementType">
1829   <complexContent>
1830     <extension base="saml:SubjectStatementAbstractType" >
1831       <sequence>
1832         <element ref="saml:AuthenticationLocality" minOccurs="0" />
1833         <element ref="saml:AuthorityBinding"
1834           minOccurs="0" maxOccurs="unbounded" />
1835       </sequence>
1836       <attribute name="AuthenticationMethod" type="anyURI" />
1837       <attribute name="AuthenticationInstant" type="dateTime" />
1838     </extension>
1839   </complexContent>
1840 </complexType>
1841 <element name="AuthenticationLocality"
1842   type="saml:AuthenticationLocalityType" />
1843 <complexType name="AuthenticationLocalityType" >
1844   <attribute name="IPAddress" type="string" use="optional" />
1845   <attribute name="DNSAddress" type="string" use="optional" />
1846 </complexType>
1847 <element name="AuthorityBinding" type="saml:AuthorityBindingType" />
1848 <complexType name="AuthorityBindingType">
1849   <attribute name="AuthorityKind" type="saml:AuthorityKindType"
1850     use="required" />
1851   <attribute name="Location" type="anyURI" use="required" />
1852   <attribute name="Binding" type="anyURI" use="required" />
1853 </complexType>
1854 <simpleType name="AuthorityKindType" >
1855   <restriction base="string" >
1856     <enumeration value="authentication" />
1857     <enumeration value="attribute" />
1858     <enumeration value="authorization" />
1859   </restriction>
1860 </simpleType>
1861 <element name="AuthorizationDecisionStatement"
1862   type="saml:AuthorizationDecisionStatementType" />
1863 <complexType name="AuthorizationDecisionStatementType" >
```

```

1864 <complexContent>
1865   <extension base="saml:SubjectStatementAbstractType">
1866     <sequence>
1867       <element ref="saml:Actions"/>
1868       <element ref="saml:Evidence"
1869         minOccurs="0" maxOccurs="unbounded"/>
1870     </sequence>
1871     <attribute name="Resource" type="anyURI" use="requiredoptional"/>
1872     <attribute name="Decision"
1873       type="saml:DecisionType" use="requiredoptional"/>
1874   </extension>
1875 </complexContent>
1876 </complexType>
1877 <element name="Actions" type="saml:ActionsType" />
1878 <complexType name="ActionsType">
1879   <sequence>
1880     <element ref="saml:Action" maxOccurs="unbounded"/>
1881   </sequence>
1882   <attribute name="Namespace" type="anyURI" use="optional" />
1883 </complexType>
1884 <element name="Action" type="string" />
1885 <element name="Evidence" type="saml:AssertionSpecifierType" />
1886 <element name="AttributeStatement" type="saml:AttributeStatementType" />
1887 <complexType name="AttributeStatementType">
1888   <complexContent>
1889     <extension base="saml:SubjectStatementAbstractType">
1890       <sequence>
1891         <element ref="saml:Attribute" maxOccurs="unbounded" />
1892       </sequence>
1893     </extension>
1894   </complexContent>
1895 </complexType>
1896 <element name="AttributeDesignator" type="saml:AttributeDesignatorType" />
1897 <complexType name="AttributeDesignatorType">
1898   <attribute name="AttributeName" type="string" use="required" />
1899   <attribute name="AttributeNamespace" type="anyURI" use="required" />
1900 </complexType>
1901 <element name="Attribute" type="saml:AttributeType" />
1902 <complexType name="AttributeType">
1903   <complexContent>
1904     <extension base="saml:AttributeDesignatorType">
1905       <sequence>
1906         <element ref="saml:AttributeValue" maxOccurs="unbounded" />
1907       </sequence>
1908     </extension>
1909   </complexContent>
1910 </complexType>
1911 <element name="AttributeValue" type="saml:anyType" />
1912 </schema>
```

8.2. Protocol Schema

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

```

1914 <?xml version="1.0" encoding="UTF-8"?>
1915 <! -- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-Baker
1916 (VeriSign Inc.) -->
1917 <schema
1918   targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-2726.xsd"
1919   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1920   xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-2726.xsd"
```

```

1924     xmlns:samlp="http://www.oasis-open.org/committees/security/docs/draft-sstc-
1925     schema-protocol-2726.xsd"
1926     xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
1927     <import
1928         namespace="http://www.oasis-open.org/committees/security/docs/draft-sstc-
1929         schema-assertion-2726.xsd"
1930         schemaLocation="draft-sstc-schema-assertion-2726.xsd"/>
1931     <import namespace="http://www.w3.org/2000/09/xmldsig#"
1932         schemaLocation="xmldsig-core-schema.xsd"/>
1933     <annotation>
1934         <documentation>draft-sstc-schema-protocol-2726.xsd</documentation>
1935     </annotation>
1936     <complexType name="RequestAbstractType" abstract="true">
1937         <sequence>
1938             <element ref="samlp:RespondWith"
1939                 minOccurs="0" maxOccurs="unbounded" />
1940             <element ref = "ds:Signature" minOccurs="0" />
1941         </sequence>
1942         <attribute name="RequestID" type="saml:IDType" use="required" />
1943         <attribute name="MajorVersion" type="integer" use="required" />
1944         <attribute name="MinorVersion" type="integer" use="required" />
1945         <attribute name="IssueInstant" type="dateTime" use="required" />
1946     </complexType>
1947     <element name="RespondWith" type="anyURI" />
1948     <element name="Request" type="samlp:RequestType" />
1949     <complexType name="RequestType">
1950         <complexContent>
1951             <extension base="samlp:RequestAbstractType">
1952                 <choice>
1953                     <element ref="samlp:Query" />
1954                     <element ref="samlp:SubjectQuery" />
1955                     <element ref="samlp:AuthenticationQuery" />
1956                     <element ref="samlp:AttributeQuery" />
1957                     <element ref="samlp:AuthorizationDecisionQuery" />
1958                     <element ref="saml:AssertionID" maxOccurs="unbounded" />
1959                     <element ref="samlp:AssertionArtifact" maxOccurs="unbounded" />
1960                 </choice>
1961             </extension>
1962         </complexContent>
1963     </complexType>
1964     <element name="AssertionArtifact" type="string" />
1965     <element name="Query" type="samlp:QueryAbstractType" />
1966     <complexType name="QueryAbstractType" abstract="true" />
1967     <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType" />
1968     <complexType name="SubjectQueryAbstractType" abstract="true" >
1969         <complexContent>
1970             <extension base="samlp:QueryAbstractType">
1971                 <sequence>
1972                     <element ref="saml:Subject" />
1973                 </sequence>
1974             </extension>
1975         </complexContent>
1976     </complexType>
1977     <element name="AuthenticationQuery" type="samlp:AuthenticationQueryType" />
1978     <complexType name="AuthenticationQueryType" >
1979         <complexContent>
1980             <extension base="samlp:SubjectQueryAbstractType" >
1981                 <sequence>
1982                     <element ref="saml:ConfirmationMethod" minOccurs="0" />
1983                 </sequence>
1984             </extension>
1985         </complexContent>
1986     </complexType>

```

```

1987 <element name="AttributeQuery" type="samlp:AttributeQueryType" />
1988 <complexType name="AttributeQueryType">
1989   <complexContent>
1990     <extension base="samlp:SubjectQueryAbstractType">
1991       <sequence>
1992         <element ref="saml:AttributeDesignator"
1993           minOccurs="0" maxOccurs="unbounded"/>
1994       </sequence>
1995       <attribute name="Resource" type="anyURI" use="optional"/>
1996     </extension>
1997   </complexContent>
1998 </complexType>
1999 <element name="AuthorizationDecisionQuery"
2000   type="samlp:AuthorizationDecisionQueryType" />
2001 <complexType name="AuthorizationDecisionQueryType">
2002   <complexContent>
2003     <extension base="samlp:SubjectQueryAbstractType">
2004       <sequence>
2005         <element ref="saml:Actions" />
2006         <element ref="saml:Evidence"
2007           minOccurs="0" maxOccurs="unbounded"/>
2008       </sequence>
2009       <attribute name="Resource" type="anyURI" use="required"/>
2010     </extension>
2011   </complexContent>
2012 </complexType>
2013 <complexType name="ResponseAbstractType" abstract="true" >
2014   <sequence>
2015     <element ref = "ds:Signature" minOccurs="0" />
2016   </sequence>
2017   <attribute name="ResponseID" type="saml:IDType" use="required" />
2018   <attribute name="InResponseTo" type="saml:IDReferenceType"
2019     use="required" />
2020   <attribute name="MajorVersion" type="integer" use="required" />
2021   <attribute name="MinorVersion" type="integer" use="required" />
2022   <attribute name="IssueInstant" type="dateTime" use="required" />
2023 </complexType>
2024
2025 <element name="Response" type="samlp:ResponseType" />
2026 <complexType name="ResponseType">
2027   <complexContent>
2028     <extension base="samlp:ResponseAbstractType">
2029       <sequence>
2030         <element ref="samlp>Status" />
2031         <element ref="saml:Assertion"
2032           minOccurs="0" maxOccurs="unbounded" />
2033       </sequence>
2034     </extension>
2035   </complexContent>
2036 </complexType>
2037 <element name="Status" type="samlp:StatusType" />
2038 <complexType name="StatusType">
2039   <sequence>
2040     <element ref="samlp:StatusCode" />
2041     <element ref="samlp:StatusMessage"
2042       minOccurs="0" maxOccurs="unbounded" />
2043     <element ref="samlp:StatusDetail" minOccurs="0" />
2044   </sequence>
2045 </complexType>
2046 <element name="StatusCode" type="samlp:StatusCodeType" />
2047 <complexType name="StatusCodeType">
2048   <sequence>
2049     <element ref="samlp:SubStatusCode" minOccurs="0" />

```

```

2050      </sequence>
2051      <attribute name="Value" type="samlp:StatusCodeEnumType" use="required" />
2052  </complexType>
2053  <simpleType name="StatusCodeEnumType " >
2054      <restriction base="QName " >
2055          <enumeration value="samlp:Success" />
2056          <enumeration value="samlp:VersionMismatch" />
2057          <enumeration value="samlp:Receiver" />
2058          <enumeration value="samlp:Sender" />
2059      </restriction>
2060  </simpleType>
2061  <element name="SubStatusCode" type="samlp:SubStatusCodeType " />
2062  <complexType name="SubStatusCodeType " >
2063      <sequence>
2064          <element ref="samlp:SubStatusCode" minOccurs="0 " />
2065      </sequence>
2066      <attribute name="Value" type="QName" use="required" />
2067  </complexType>
2068  <element name="StatusMessage" type="string" />
2069  <element name="StatusDetail" type="samlp:StatusDetailType" />
2070  <complexType name="StatusDetailType" >
2071      <sequence>
2072          <any namespace="##any"
2073              processContents="lax" minOccurs="0" maxOccurs="unbounded" />
2074      </sequence>
2075  </complexType>
2076</schema>
2077
```

2078 9. References

- 2079 [Needham78] R. Needham et al., *Using Encryption for Authentication in Large Networks*
2080 of Computers
- 2081 Communications of the ACM, Vol. 21 (12), pp. 993-999,
December 1978.
- 2082 [Kern-84] B. Kernighan, Rob Pike *The UNIX Programming Environment*, (March
2083 1984) Prentice Hall Computer Books;
- 2084 [PKCS1] B. Kaliski, *PKCS #1: RSA Encryption Version 2.0*, RSA Laboratories, also
2085 IETF RFC 2437, October 1998. <http://www.ietf.org/rfc/rfc2437.txt>
- 2086 [PKCS7] B. Kaliski., "PKCS #7: Cryptographic Message Syntax, Version 1.5.", RFC
2087 2315, March 1998.
- 2088 [RFC 1510] J. Kohl, C. Neuman. *The Kerberos Network Authentication Requestor (V5)*.
2089 September 1993. <http://www.ietf.org/rfc/rfc1510.txt>
- 2090 [RFC 2246] T. Dierks, C. Allen. *The TLS Protocol Version 1.0*. January 1999.
2091 <http://www.ietf.org/rfc/rfc2246.txt>
- 2092 [RFC 2630] R. Housley. Cryptographic Message Syntax. June 1999.
2093 <http://www.ietf.org/rfc/rfc630.txt>
- 2094 [RFC 2648] R. Moats. A *URN Namespace for IETF Documents*. August 1999.
2095 <http://www.ietf.org/rfc/rfc2648.txt>
- 2096 [RFC 3075] D. Eastlake, J. Reagle, D. Solo. XML-Signature Syntax and Processing.
2097 March 2001. <http://www.ietf.org/rfc/rfc3075.txt>
- 2098 [RFC2104] H. Krawczyk et al., *HMAC: Keyed Hashing for Message Authentication*,
2099 <http://www.ietf.org/rfc/rfc2104.txt>, IETF RFC 2104, February 1997.
- 2100 [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
2101 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997
- 2102 [SAMLBind] P. Mishra et al., *Bindings and Profiles for the OASIS Security Assertion
2103 Markup Language (SAML)*, [http://www.oasis-
2104 open.org/committees/security/docs/draft-sstc-bindings-model-07.pdf](http://www.oasis-open.org/committees/security/docs/draft-sstc-bindings-model-07.pdf), OASIS,
2105 December 2001.
- 2106 [SAMLConform] **TBS**
- 2107 [SAMLGloss] J. Hodges et al., *Glossary for the OASIS Security Assertion Markup
2108 Language (SAML)*, [http://www.oasis-
2109 open.org/committees/security/docs/draft-sstc-glossary-02.pdf](http://www.oasis-open.org/committees/security/docs/draft-sstc-glossary-02.pdf), OASIS,
2110 December 2001.
- 2111 [SAMPL-XSD] P. Hallam-Baker et al., *SAML protocol schema*, [http://www.oasis-
2112 open.org/committees/security/docs/draft-sstc-schema-protocol-21.xsd](http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-21.xsd), OASIS,
2113 December 2001.
- 2114 [SAMLSecure] **TBS**
- 2115 [SAML-XSD] P. Hallam-Baker et al., *SAML assertion schema*, [http://www.oasis-
2116 open.org/committees/security/docs/draft-sstc-schema-assertion-21.xsd](http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-21.xsd), OASIS,
2117 December 2001.
- 2118 [Schema1] H. S. Thompson et al., *XML Schema Part 1: Structures*,
2119 <http://www.w3.org/TR/xmlschema-1/>, World Wide Web Consortium
2120 Recommendation, May 2001.
- 2121 [Schema2] P. V. Biron et al., *XML Schema Part 2: Datatypes*,
2122 <http://www.w3.org/TR/xmlschema-2>, World Wide Web Consortium
2123 Recommendation, May 2001.
- 2124 [XMLEnc] *XML Encryption Specification*, In development.

2125	[XMLSig]	D. Eastlake et al., <i>XML-Signature Syntax and Processing</i> , http://www.w3.org/TR/xmldsig-core/ , World Wide Web Consortium.
2126		
2127	[XMLSig-XSD]	XML Signature Schema available from http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/xmldsig-core-schema.xsd
2128		
2129	[XTAML]	P. Hallam-Baker, <i>XML Trust Axiom Markup Language 1.0</i> , http://www.xmltrustcenter.org/ , VeriSign Inc. September 2001.
2130		
2131	[W3C-CHAR]	http://www.w3.org/TR/WD-charreq
2132	[UNICODE-C]	http://www.unicode.org/unicode/reports/tr15/tr15-21.html
2133	[W3C-CharMod]	http://www.w3.org/TR/charmod/
2134	[XML]	http://www.w3.org/TR/REC-xml
2135	[RFC 2396]	http://www.ietf.org/rfc/rfc2396.txt?

2136 Appendix A. Notices

2137 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
2138 that might be claimed to pertain to the implementation or use of the technology described in this
2139 document or the extent to which any license under such rights might or might not be available;
2140 neither does it represent that it has made any effort to identify any such rights. Information on
2141 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
2142 website. Copies of claims of rights made available for publication and any assurances of licenses to
2143 be made available, or the result of an attempt made to obtain a general license or permission for
2144 the use of such proprietary rights by implementors or users of this specification, can be obtained
2145 from the OASIS Executive Director.

2146 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
2147 applications, or other proprietary rights which may cover technology that may be required to
2148 implement this specification. Please address the information to the OASIS Executive Director.

2149 Copyright © The Organization for the Advancement of Structured Information Standards [OASIS]
2150 2001. All Rights Reserved.

2151 This document and translations of it may be copied and furnished to others, and derivative works
2152 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
2153 published and distributed, in whole or in part, without restriction of any kind, provided that the above
2154 copyright notice and this paragraph are included on all such copies and derivative works. However,
2155 this document itself may not be modified in any way, such as by removing the copyright notice or
2156 references to OASIS, except as needed for the purpose of developing OASIS specifications, in
2157 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights
2158 document must be followed, or as required to translate it into languages other than English.

2159 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
2160 successors or assigns.

2161 This document and the information contained herein is provided on an "AS IS" basis and OASIS
2162 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
2163 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY
2164 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
2165 PARTICULAR PURPOSE.