



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

Document identifier: cs-sstc-core-00

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

Publication date: 19 April 2002

Maturity level: Committee Specification

Send comments to: If you are on the security-services@lists.oasis-open.org list for committee members, send comments there. If you are not on that list, subscribe to the security-services-comment@lists.oasis-open.org list and send comments there. To subscribe, send an email message to security-services-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

Editors:

Phillip Hallam-Baker, VeriSign (pbaker@verisign.com)

Eve Maler, Sun Microsystems (eve.maler@sun.com)

Contributors:

Stephen Farrell, Baltimore Technologies

Irving Reid, Baltimore Technologies

David Orchard, BEA Systems

Krishna Sankar, Cisco Systems

Simon Godik, Crosslogix

Hal Lockhart, Entegritiy

Carlisle Adams, Entrust

Tim Moses, Entrust

Nigel Edwards, Hewlett-Packard

Joe Pato, Hewlett-Packard

Marc Chanliau, Netegrity

Chris McLaren, Netegrity

Prateek Mishra, Netegrity

Charles Knouse, Oblix

Scott Cantor, Ohio State University

Darren Platt, formerly with RSA Security

Jeff Hodges, Sun Microsystems

Bob Blakley Tivoli

Marlena Erdos, Tivoli

RL "Bob" Morgan, University of Washington

37		
38	Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)	1
39	1 Introduction	5
40	1.1 Notation	5
41	1.2 Schema Organization and Namespaces	5
42	1.2.1 String and URI Values	6
43	1.2.2 Time Values	6
44	1.2.3 Comparing SAML values	6
45	1.3 SAML Concepts (Non-Normative)	6
46	1.3.1 Overview	6
47	1.3.2 SAML and URI-Based Identifiers	8
48	1.3.3 SAML and Extensibility	8
49	2 SAML Assertions	9
50	2.1 Schema Header and Namespace Declarations	9
51	2.2 Simple Types	9
52	2.2.1 Simple Types IDType and IDReferenceType	9
53	2.2.2 Simple Type DecisionType	10
54	2.3 Assertions	10
55	2.3.1 Element <AssertionIDReference>	10
56	2.3.2 Element <Assertion>	10
57	2.4 Statements	14
58	2.4.1 Element <Statement>	14
59	2.4.2 Element <SubjectStatement>	14
60	2.4.3 Element <AuthenticationStatement>	17
61	2.4.4 Element <AuthorizationDecisionStatement>	18
62	2.4.5 Element <AttributeStatement>	21
63	3 SAML Protocol	23
64	3.1 Schema Header and Namespace Declarations	23
65	3.2 Requests	23
66	3.2.1 Complex Type RequestAbstractType	23
67	3.2.2 Element <Request>	24
68	3.2.3 Element <AssertionArtifact>	25
69	3.3 Queries	25
70	3.3.1 Element <Query>	25
71	3.3.2 Element <SubjectQuery>	26
72	3.3.3 Element <AuthenticationQuery>	26
73	3.3.4 Element <AttributeQuery>	27
74	3.3.5 Element <AuthorizationDecisionQuery>	27
75	3.4 Responses	28

76	3.4.1 Complex Type ResponseAbstractType	28
77	3.4.2 Element <Response>	29
78	3.4.3 Element <Status>	29
79	3.4.4 Responses to <AuthenticationQuery> and <AttributeQuery>	31
80	4 SAML Versioning	33
81	4.1 Assertion Version	33
82	4.2 Request Version.....	33
83	4.3 Response Version.....	33
84	5 SAML and XML Signature Syntax and Processing	35
85	5.1 Signing Assertions	35
86	5.2 Request/Response Signing	36
87	5.3 Signature Inheritance.....	36
88	5.3.1 Rationale.....	36
89	5.3.2 Rules for SAML Signature Inheritance	36
90	5.4 XML Signature Profile	36
91	5.4.1 Signing Formats.....	36
92	5.4.2 CanonicalizationMethod	36
93	5.4.3 Transforms.....	37
94	5.4.4 KeyInfo.....	37
95	5.4.5 Binding Between Statements in a Multi-Statement Assertion	37
96	6 SAML Extensions	38
97	6.1 Assertion Schema Extension	38
98	6.2 Protocol Schema Extension.....	38
99	6.3 Use of Type Derivation and Substitution Groups.....	39
100	7 SAML-Defined Identifiers.....	40
101	7.1 Authentication Method Identifiers.....	40
102	7.1.1 Password :	40
103	7.1.2 Kerberos	40
104	7.1.3 Secure Remote Password (SRP)	40
105	7.1.4 Hardware Token	41
106	7.1.5 SSL/TLS Certificate Based Client Authentication:.....	41
107	7.1.6 X.509 Public Key	41
108	7.1.7 PGP Public Key	41
109	7.1.8 SPKI Public Key.....	41
110	7.1.9 XKMS Public Key.....	41
111	7.1.10 XML Digital Signature	41
112	7.2 Action Namespace Identifiers	42
113	7.2.1 Read/Write/Execute/Delete/Control:.....	42

114	7.2.2 Read/Write/Execute/Delete/Control with Negation:.....	42
115	7.2.3 Get/Head/Put/Post:.....	42
116	7.2.4 UNIX File Permissions:.....	43
117	8 SAML Schema Listings.....	44
118	8.1 Assertion Schema.....	44
119	8.2 Protocol Schema.....	48
120	9 References.....	51
121	Appendix A. Acknowledgments.....	53
122	Appendix B. Notices.....	54
123		

1 Introduction

This specification defines the syntax and semantics for XML-encoded SAML assertions, protocol requests, and protocol responses. These constructs are typically embedded in other structures for transport, such as HTTP form POSTs and XML-encoded SOAP messages. The SAML specification for bindings and profiles **[SAMLBind]** provides frameworks for this embedding and transport. Files containing just the SAML assertion schema **[SAML-XSD]** and protocol schema **[SAML-P-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 **[RFC 2119]**:

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

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

Listings of SAML schemas appear like this.

Example code listings appear like this.

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

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

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

1.2 Schema Organization and Namespaces

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

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

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

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

Note: The SAML namespace names may change when SAML 1.0 becomes an OASIS Standard.

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

168 **1.2.1 String and URI Values**

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

174 **1.2.2 Time Values.**

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

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

179 **1.2.3 Comparing SAML values**

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

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

192 Applications that compare data received in SAML documents to data from external sources **MUST** take
193 into account the normalization rules specified for XML. Text contained within elements is normalized so
194 that line endings are represented using linefeed characters (ASCII code 10_{Decimal}), as described in section
195 2.11 of the XML Recommendation [**XML**]. Attribute values defined as strings (or types derived from
196 strings) are normalized as described in section 3.3.3 [**XML**]. All white space characters are replaced with
197 blanks (ASCII code 32_{Decimal}).

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

201 **1.3 SAML Concepts (Non-Normative)**

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

204 **1.3.1 Overview**

205 The Security Assertion Markup Language (SAML) is an XML-based framework for exchanging security
206 information. This security information is expressed in the form of assertions about subjects, where a

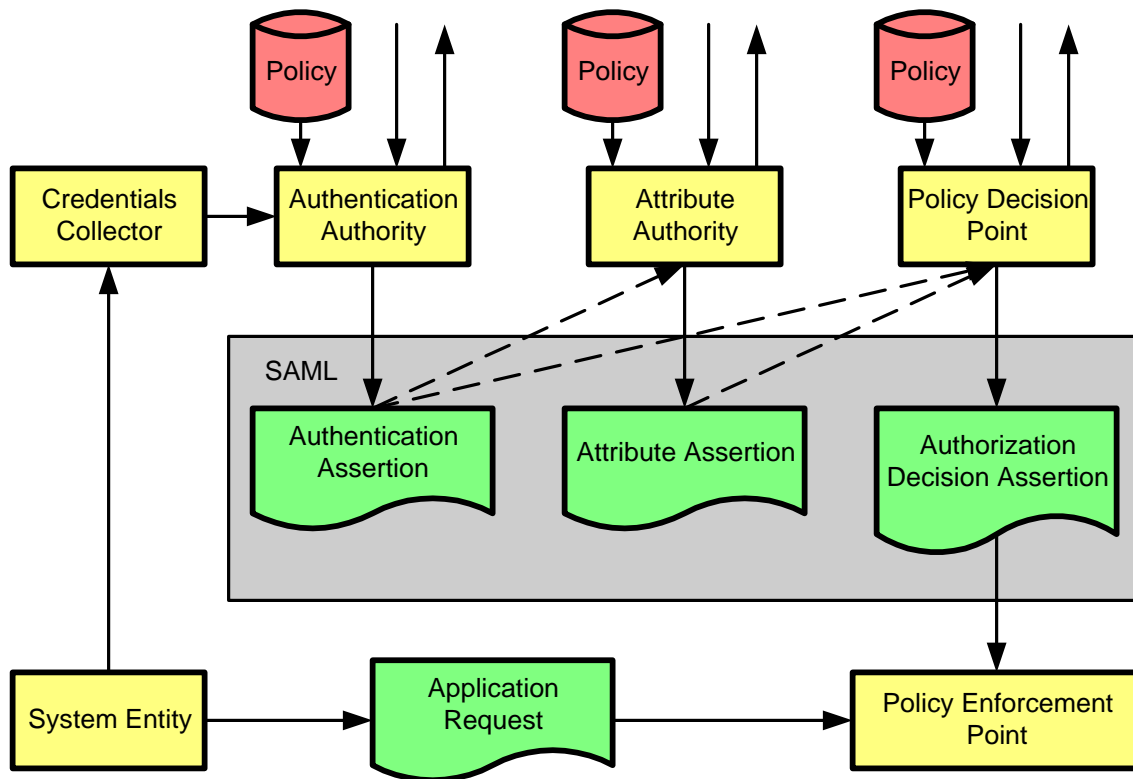
207 subject is an entity (either human or computer) that has an identity in some security domain. A typical
208 example of a subject is a person, identified by his or her email address in a particular Internet DNS
209 domain.

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

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

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

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



226

227

Figure 1 The SAML Domain Model

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

232 The assertion and protocol data formats are defined in this specification. The bindings and profiles are
233 defined in a separate specification **[SAMLBind]**. A conformance program for SAML is defined in the

234 conformance specification [**SAMLConform**]. Security issues are discussed in a separate security and
235 privacy considerations specification [**SAMLSecure**].

236 **1.3.2 SAML and URI-Based Identifiers**

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

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

240

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

243 **urn:oasis:names:tc:SAML:1.0:action:rwedc**

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

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

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

254 **1.3.3 SAML and Extensibility**

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

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

258 2 SAML Assertions

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

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

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

269 2.1 Schema Header and Namespace Declarations

270 The following schema fragment defines the XML namespaces and other header information for the
271 assertion schema:

```
272 <schema  
273   targetNamespace="urn:oasis:names:tc:SAML:1.0:assertion"  
274   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
275   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"  
276   xmlns="http://www.w3.org/2001/XMLSchema"  
277   elementFormDefault="unqualified">  
278   <import namespace="http://www.w3.org/2000/09/xmldsig#"  
279     schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-  
280 schema.xsd"/>  
281   <annotation>  
282     <documentation>cs-sstc-schema-assertion-00</documentation>  
283   </annotation>  
284   ...  
285 </schema>
```

286 2.2 Simple Types

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

288 2.2.1 Simple Types *IDType* and *IDReferenceType*

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

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

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

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

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

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

```
305     <simpleType name="IDType">  
306         <restriction base="string"/>  
307     </simpleType>  
308     <simpleType name="IDReferenceType">  
309         <restriction base="string"/>  
310     </simpleType>
```

311 **2.2.2 Simple Type DecisionType**

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

314 Permit

315 The specified action is permitted.

316 Deny

317 The specified action is denied.

318 Indeterminate

319 The issuer cannot determine whether the specified action is permitted or denied.

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

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

```
324     <simpleType name="DecisionType">  
325         <restriction base="string">  
326             <enumeration value="Permit"/>  
327             <enumeration value="Deny"/>  
328             <enumeration value="Indeterminate"/>  
329         </restriction>  
330     </simpleType>
```

331 **2.3 Assertions**

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

333 **2.3.1 Element <AssertionIDReference>**

334 The <AssertionIDReference> element makes a reference to a SAML assertion.

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

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

337 **2.3.2 Element <Assertion>**

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

340 MajorVersion [Required]

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

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

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

349 Issuer [Required]
 350 The issuer of the assertion. The name of the issuer is provided as a string. The issuer name
 351 **SHOULD** be unambiguous to the intended relying parties. SAML authorities may use an identifier
 352 such as a URI reference that is designed to be unambiguous regardless of context.

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

355 <Conditions> [Optional]
 356 Conditions that **MUST** be taken into account in assessing the validity of the assertion.

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

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

362 One or more of the following statement elements:

363 <Statement>
 364 A statement defined in an extension schema.

365 <SubjectStatement>
 366 A subject statement defined in an extension schema.

367 <AuthenticationStatement>
 368 An authentication statement.

369 <AuthorizationDecisionStatement>
 370 An authorization decision statement.

371 <AttributeStatement>
 372 An attribute statement.

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

```

374     <element name="Assertion" type="saml:AssertionType"/>
375     <complexType name="AssertionType">
376         <sequence>
377             <element ref="saml:Conditions" minOccurs="0"/>
378             <element ref="saml:Advice" minOccurs="0"/>
379             <choice maxOccurs="unbounded">
380                 <element ref="saml:Statement"/>
381                 <element ref="saml:SubjectStatement"/>
382                 <element ref="saml:AuthenticationStatement"/>
383                 <element ref="saml:AuthorizationDecisionStatement"/>
384                 <element ref="saml:AttributeStatement"/>
385             </choice>
386             <element ref="ds:Signature" minOccurs="0"/>
387         </sequence>
388         <attribute name="MajorVersion" type="integer" use="required"/>
389         <attribute name="MinorVersion" type="integer" use="required"/>
390         <attribute name="AssertionID" type="saml:IDType" use="required"/>
391         <attribute name="Issuer" type="string" use="required"/>
392         <attribute name="IssueInstant" type="dateTime" use="required"/>
393     </complexType>

```

394 **2.3.2.1 Element <Conditions>**

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

396 `NotBefore` [Optional]

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

399 `NotOnOrAfter` [Optional]

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

402 <Condition> [Any Number]

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

404 <AudienceRestrictionCondition> [Any Number]

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

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

```
408 <element name="Conditions" type="saml:ConditionsType"/>  
409 <complexType name="ConditionsType">  
410 <choice minOccurs="0" maxOccurs="unbounded">  
411 <element ref="saml:AudienceRestrictionCondition"/>  
412 <element ref="saml:Condition"/>  
413 </choice>  
414 <attribute name="NotBefore" type="dateTime" use="optional"/>  
415 <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>  
416 </complexType>
```

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

- 421 1. If no sub-elements or attributes are supplied in the <Conditions> element, then the assertion
422 is considered to be **Valid**.
- 423 2. If any sub-element or attribute of the <Conditions> element is determined to be invalid, then
424 the assertion is **Invalid**.
- 425 3. If any sub-element or attribute of the <Conditions> element cannot be evaluated, then the
426 validity of the assertion cannot be determined and is deemed to be **Indeterminate**.
- 427 4. If all sub-elements and attributes of the <Conditions> element are determined to be **Valid**,
428 then the assertion is considered to be **Valid**.

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

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

435 **2.3.2.1.1 Attributes `NotBefore` and `NotOnOrAfter`**

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

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

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

439 If the value for either `NotBefore` or `NotOnOrAfter` is omitted it is considered unspecified. If the
440 `NotBefore` attribute is unspecified (and if any other conditions that are supplied evaluate to `Valid`), the
441 assertion is valid at any time before the time instant specified by the `NotOnOrAfter` attribute. If the
442 `NotOnOrAfter` attribute is unspecified (and if any other conditions that are supplied evaluate to `Valid`),
443 the assertion is valid from the time instant specified by the `NotBefore` attribute with no expiry. If neither
444 attribute is specified (and if any other conditions that are supplied evaluate to `Valid`), the assertion is
445 valid at any time.

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

450 **2.3.2.1.2 Element <Condition>**

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

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

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

458 **2.3.2.1.3 Elements <AudienceRestrictionCondition> and <Audience>**

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

463 `<Audience>`

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

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

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

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

```
475 <element name="AudienceRestrictionCondition"  
476         type="saml:AudienceRestrictionConditionType"/>  
477 <complexType name="AudienceRestrictionConditionType">  
478   <complexContent>  
479     <extension base="saml:ConditionAbstractType">  
480       <sequence>  
481         <element ref="saml:Audience"  
482 maxOccurs="unbounded"/>  
483       </sequence>  
484     </extension>  
485   </complexContent>
```

```
486     </complexType>
487     <element name="Audience" type="anyURI" />
```

488 2.3.2.2 Elements <Advice> and <AdviceElement>

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

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

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

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

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

```
501     <element name="Advice" type="saml:AdviceType" />
502     <complexType name="AdviceType">
503         <choice minOccurs="0" maxOccurs="unbounded">
504             <element ref="saml:AssertionIDReference" />
505             <element ref="saml:Assertion" />
506             <any namespace="##other" processContents="lax" />
507         </choice>
508     </complexType>
```

509 2.4 Statements

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

511 2.4.1 Element <Statement>

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

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

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

519 2.4.2 Element <SubjectStatement>

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

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

```

527     <element name="SubjectStatement"
528 type="saml:SubjectStatementAbstractType" />
529     <complexType name="SubjectStatementAbstractType" abstract="true">
530       <complexContent>
531         <extension base="saml:StatementAbstractType">
532           <sequence>
533             <element ref="saml:Subject" />
534           </sequence>
535         </extension>
536       </complexContent>
537     </complexType>

```

538 2.4.2.1 Element <Subject>

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

541 <NameIdentifier>

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

543 <SubjectConfirmation>

544 Information that allows the subject to be authenticated.

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

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

```

550     <element name="Subject" type="saml:SubjectType" />
551     <complexType name="SubjectType">
552       <choice>
553         <sequence>
554           <element ref="saml:NameIdentifier" />
555           <element ref="saml:SubjectConfirmation"
556 minOccurs="0" />
557         </sequence>
558         <element ref="saml:SubjectConfirmation" />
559       </choice>
560     </complexType>

```

561 2.4.2.2 Element <NameIdentifier>

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

564 NameQualifier [Optional]

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

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

568 Format [Optional]

569 The syntax used to describe the name of the subject

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

573 #emailAddress

574 Indicates that the content of the NameIdentifier element is in the form of an email address,
575 specifically "addr-spec" as defined in section 3.4.1 of RFC 2822 [RFC 2822]. An addr-spec has

576 the form local-part@domain. Note that an addr-spec has no phrase (such as a common name)
577 before it, has no comment (text surrounded in parentheses) after it, and is not surrounded by "<"
578 and ">".

579 #X509SubjectName

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

583 #WindowsDomainQualifiedName

584 Indicates that the content of the NameIdentifier element is a Windows domain qualified name. A
585 Windows domain qualified user name is a string of the form "DomainName\UserName". The
586 domain name and "\" separator may be omitted.

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

```
589 <element name="NameIdentifier" type="saml:NameIdentifierType"/>  
590 <complexType name="NameIdentifierType">  
591 <simpleContent>  
592 <extension base="string">  
593 <attribute name="NameQualifier" type="string"  
594 use="optional"/>  
595 <attribute name="Format" type="anyURI "  
596 use="optional"/>  
597 </extension>  
598 </simpleContent>  
599 </complexType>
```

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

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

604 **2.4.2.3 Elements <SubjectConfirmation>, <ConfirmationMethod>, 605 and <SubjectConfirmationData>**

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

608 <ConfirmationMethod> [One or more]

609 A URI reference that identifies a protocol to be used to authenticate the subject. URI references
610 identifying SAML-defined confirmation methods are currently defined with the SAML profiles in
611 **[SAMLBind]**. Additional methods may be added by defining new profiles or by private
612 agreement.

613 <SubjectConfirmationData> [Optional]

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

615 <ds:KeyInfo> [Optional]

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

617 The following schema fragment defines the <SubjectConfirmation> element and its

618 **SubjectConfirmationType** complex type, along with the <SubjectConfirmationData> element and
619 the <ConfirmationMethod> element:

```
620 <element name="SubjectConfirmation"  
621 type="saml:SubjectConfirmationType"/>  
622 <complexType name="SubjectConfirmationType">  
623 <sequence>
```



```

624         <element ref="saml:ConfirmationMethod"
625 maxOccurs="unbounded" />
626         <element ref="saml:SubjectConfirmationData" minOccurs="0" />
627         <element ref="ds:KeyInfo" minOccurs="0" />
628     </sequence>
629 </complexType>
630 <element name="SubjectConfirmationData" type="anyType" />
631 <element name="ConfirmationMethod" type="anyURI" />

```

632 2.4.3 Element <AuthenticationStatement>

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

636 AuthenticationMethod [Optional]

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

639 AuthenticationInstant [Optional]

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

642 <SubjectLocality> [Optional]

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

645 <AuthorityBinding> [Any Number]

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

647 The following schema fragment defines the <AuthenticationStatement> element and its

648 **AuthenticationStatementType** complex type:

```

649     <element name="AuthenticationStatement"
650           type="saml:AuthenticationStatementType" />
651     <complexType name="AuthenticationStatementType">
652         <complexContent>
653             <extension base="saml:SubjectStatementAbstractType">
654                 <sequence>
655                     <element ref="saml:SubjectLocality"
656 minOccurs="0" />
657                     <element ref="saml:AuthorityBinding"
658 minOccurs="0" maxOccurs="unbounded" />
659                 </sequence>
660                 <attribute name="AuthenticationMethod"
661 type="anyURI" />
662                 <attribute name="AuthenticationInstant"
663 type="dateTime" />
664             </extension>
665         </complexContent>
666     </complexType>

```

667 2.4.3.1 Element <SubjectLocality>

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

670 IPAddress [Optional]

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

672 DNSAddress [Optional]
673 The DNS address of the system entity that was authenticated.
674 This element is entirely advisory, since both these fields are quite easily “spoofed” but current practice
675 appears to require its inclusion.
676 The following schema fragment defines the <SubjectLocality> element and its **SubjectLocalityType**
677 complex type:

```
678     <element name="SubjectLocality"  
679             type="saml: SubjectLocalityType" />  
680     <complexType name="SubjectLocalityType">  
681         <attribute name="IPAddress" type="string" use="optional" />  
682         <attribute name="DNSAddress" type="string" use="optional" />  
683     </complexType>
```

684 2.4.3.2 Element <AuthorityBinding>

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

690 AuthorityKind [Required]

691 The type of SAML Protocol queries to which the authority described by this element will respond.
692 The value is specified as an XML Schema QName. The acceptable values for AuthorityKind
693 are the namespace-qualified names of element types or elements derived from the SAML
694 Protocol Query element (see Section 3.3). For example, an attribute authority would be identified
695 by AuthorityKind="samlp:AttributeQuery". For extension schemas, where the actual
696 type of the <samlp:Query> would be identified by an xsi:type attribute, the value of
697 AuthorityKind MUST be the same as the value of the xsi:type attribute for the
698 corresponding query.

699 Location [Required]

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

703 Binding [Required]

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

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

```
708     <element name="AuthorityBinding" type="saml:AuthorityBindingType" />  
709     <complexType name="AuthorityBindingType">  
710         <attribute name="AuthorityKind" type="QName" use="required" />  
711         <attribute name="Location" type="anyURI" use="required" />  
712         <attribute name="Binding" type="anyURI" use="required" />  
713     </complexType>
```

714 2.4.4 Element <AuthorizationDecisionStatement>

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

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

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

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

- 730 • The assertion issuer SHOULD encode all resource URIs in normalized form.
- 731 • Relying parties SHOULD convert resource URIs to normalized form prior to processing.

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

- 736 • Parts of the URI syntax are case sensitive. If the underlying file system is case insensitive a
737 requestor SHOULD NOT be able to gain access to a denied resource by changing the case of a
738 part of the resource URI.
- 739 • Many file systems support mechanisms such as logical paths and symbolic links which allow
740 users to establish logical equivalences between file system entries. A requestor SHOULD NOT
741 be able to gain access to a denied resource by creating such an equivalence.

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

745 Resource [Required]

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

749 Decision [Required]

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

752 <Action> [One or more]

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

754 <Evidence> [Any Number]

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

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

```
758     <element name="AuthorizationDecisionStatement"
759 type="saml:AuthorizationDecisionStatementType" />
760     <complexType name="AuthorizationDecisionStatementType">
761         <complexContent>
762             <extension base="saml:SubjectStatementAbstractType">
763                 <sequence>
764                     <element ref="saml:Action"
765 maxOccurs="unbounded" />
766                     <element ref="saml:Evidence" minOccurs="0" />
767                 </sequence>
```

```

768         <attribute name="Resource" type="anyURI"
769 use="required"/>
770         <attribute name="Decision" type="saml:DecisionType"
771 use="required"/>
772     </extension>
773 </complexContent>
774 </complexType>

```

775 2.4.4.1 Element <Action>

776 The <Action> element specifies an action on the specified resource for which permission is sought. It
777 has the following attribute and string-data content:

778 **Namespace** [Optional]

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

782 **string data** [Required]

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

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

```

785     <element name="Action" type="saml:ActionType"/>
786     <complexType name="ActionType">
787         <simpleContent>
788             <extension base="string">
789                 <attribute name="Namespace" type="anyURI"/>
790             </extension>
791         </simpleContent>
792     </complexType>

```

793 2.4.4.2 Element <Evidence>

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

796 <AssertionIDReference>

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

798 <Assertion>

799 Specifies an assertion by value.

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

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

```

805     <element name="Evidence" type="saml:EvidenceType"/>
806     <complexType name="EvidenceType">
807         <choice maxOccurs="unbounded">
808             <element ref="saml:AssertionIDReference"/>
809             <element ref="saml:Assertion"/>
810         </choice>
811     </complexType>

```

812 **2.4.5 Element <AttributeStatement>**

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

816 <Attribute> [One or More]

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

818 The following schema fragment defines the <AttributeStatement> element and its

819 **AttributeStatementType** complex type:

```
820     <element name="AttributeStatement" type="saml:AttributeStatementType"/>
821     <complexType name="AttributeStatementType">
822         <complexContent>
823             <extension base="saml:SubjectStatementAbstractType">
824                 <sequence>
825                     <element ref="saml:Attribute"
826 maxOccurs="unbounded"/>
827                 </sequence>
828             </extension>
829         </complexContent>
830     </complexType>
```

831 **2.4.5.1 Elements <AttributeDesignator> and <Attribute>**

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

836 AttributeNamespace [Optional]

837 The namespace in which the AttributeName elements are interpreted.

838 AttributeName [Optional]

839 The name of the attribute.

840 The following schema fragment defines the <AttributeDesignator> element and its

841 **AttributeDesignatorType** complex type:

```
842     <element name="AttributeDesignator"
843 type="saml:AttributeDesignatorType"/>
844     <complexType name="AttributeDesignatorType">
845         <attribute name="AttributeName" type="string" use="required"/>
846         <attribute name="AttributeNamespace" type="anyURI"
847 use="required"/>
848     </complexType>
```

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

852 <AttributeValue> [Any Number]

853 The value of the attribute.

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

```
855     <element name="Attribute" type="saml:AttributeType"/>
856     <complexType name="AttributeType">
857         <complexContent>
858             <extension base="saml:AttributeDesignatorType">
```

```
859         <sequence>
860             <element ref="saml:AttributeValue"
861 maxOccurs="unbounded" />
862         </sequence>
863     </extension>
864 </complexContent>
865 </complexType>
```

866 **2.4.5.1.1 Element <AttributeValue>**

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

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

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

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

3 SAML Protocol

875

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

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



882

883

Figure 2: SAML Request-Response Protocol

3.1 Schema Header and Namespace Declarations

884

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

```
887 <schema  
888     targetNamespace="urn:oasis:names:tc:SAML:1.0:protocol "  
889     xmlns="http://www.w3.org/2001/XMLSchema "  
890     xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol "  
891     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion "  
892     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"   
893     elementFormDefault="unqualified">  
894     <import namespace="urn:oasis:names:tc:SAML:1.0:assertion "  
895           schemaLocation="cs-sstc-schema-assertion-00.xsd"/>  
896     <import namespace="http://www.w3.org/2000/09/xmldsig#"   
897           schemaLocation="xmldsig-core-schema.xsd"/>  
898     <annotation>  
899       <documentation>cs-sstc-schema-protocol-00.xsd</documentation>  
900     </annotation>  
901     ...  
902 </schema>
```

3.2 Requests

903

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

3.2.1 Complex Type RequestAbstractType

905

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

908 RequestID [Required]

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

912 MajorVersion [Required]

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

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

918 IssueInstant [Required]
 919 The time instant of issue of the request. The time value is encoded in UTC as described in
 920 section 1.2.1.

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

923 <Signature> [Optional]
 924 An XML Signature that authenticates the assertion, see Section 5.

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

```

926     <complexType name="RequestAbstractType" abstract="true">
927         <sequence>
928             <element ref="samlp:RespondWith"
929                 minOccurs="0" maxOccurs="unbounded"/>
930             <element ref="ds:Signature" minOccurs="0"/>
931         </sequence>
932         <attribute name="RequestID" type="saml:IDType" use="required"/>
933         <attribute name="MajorVersion" type="integer" use="required"/>
934         <attribute name="MinorVersion" type="integer" use="required"/>
935         <attribute name="IssueInstant" type="dateTime" use="required"/>
936     </complexType>
  
```

937 3.2.1.1 Element <RespondWith>

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

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

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

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

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

```

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

955 3.2.2 Element <Request>

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

960 <Query>
 961 An extension point that allows extension schemas to define new types of query.
 962 <SubjectQuery>
 963 An extension point that allows extension schemas to define new types of query that specify a
 964 single SAML subject.
 965 <AuthenticationQuery>
 966 Makes a query for authentication information.
 967 <AttributeQuery>
 968 Makes a query for attribute information.
 969 <AuthorizationDecisionQuery>
 970 Makes a query for an authorization decision.
 971 <AssertionIDReference> [One or more]
 972 Requests assertions by reference to its assertion identifier.
 973 <AssertionArtifact> [One or more]
 974 Requests assertions by supplying an assertion artifact that represents it.
 975 The following schema fragment defines the <Request> element and its **RequestType** complex type:

```

976     <element name="Request" type="samlp:RequestType" />
977     <complexType name="RequestType">
978         <complexContent>
979             <extension base="samlp:RequestAbstractType">
980                 <choice>
981                     <element ref="samlp:Query" />
982                     <element ref="samlp:SubjectQuery" />
983                     <element ref="samlp:AuthenticationQuery" />
984                     <element
985 ref="samlp:AuthorizationDecisionQuery" />
986                     <element ref="saml:AssertionIDReference"
987 maxOccurs="unbounded" />
988                     <element ref="samlp:AssertionArtifact"
989 maxOccurs="unbounded" />
990                 </choice>
991             </extension>
992         </complexContent>
993     </complexType>
994 
```

995 **3.2.3 Element <AssertionArtifact>**

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

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

```

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

```

1000 **3.3 Queries**

1001 The following sections define the SAML constructs that contain query information.

1002 **3.3.1 Element <Query>**

1003 The <Query> element is an extension point that allows new SAML queries to be defined. Its
 1004 **QueryAbstractType** is abstract; extension elements **MUST** use the `xsi:type` attribute to indicate the
 1005 derived type. **QueryAbstractType** is the base type from which all SAML query elements are derived.

1006 The following schema fragment defines the <Query> element and its **QueryAbstractType** complex type:

```
1007 <element name="Query" type="samlp:QueryAbstractType" />
1008 <complexType name="QueryAbstractType" abstract="true" />
```

1009 **3.3.2 Element <SubjectQuery>**

1010 The <SubjectQuery> element is an extension point that allows new SAML queries that specify a single
1011 SAML subject. Its **SubjectQueryAbstractType** complex type is abstract; extension elements MUST use
1012 the xsi:type attribute to indicate the derived type. **SubjectQueryAbstractType** adds the <Subject>
1013 element.

1014 The following schema fragment defines the <SubjectQuery> element and its
1015 **SubjectQueryAbstractType** complex type:

```
1016 <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType" />
1017 <complexType name="SubjectQueryAbstractType" abstract="true">
1018 <complexContent>
1019 <extension base="samlp:QueryAbstractType">
1020 <sequence>
1021 <element ref="saml:Subject" />
1022 </sequence>
1023 </extension>
1024 </complexContent>
1025 </complexType>
```

1026 **3.3.3 Element <AuthenticationQuery>**

1027 The <AuthenticationQuery> element is used to make the query "What assertions containing
1028 authentication statements are available for this subject?" A successful response will be in the form of
1029 assertions containing authentication statements.

1030 Note: The <AuthenticationQuery> MAY NOT be used as a request for a new authentication using
1031 credentials provided in the request. The <AuthenticationQuery> is a request for statements about
1032 authentication acts which have occurred in a previous interaction between the indicated principal and the
1033 Authentication Authority.

1034 This element is of type **AuthenticationQueryType**, which extends **SubjectQueryAbstractType** with the
1035 addition of the following element:

1036 <AuthenticationMethod> [Optional]

1037 A filter for possible responses. If it is present, the query made is "What assertions containing
1038 authentication statements do you have for this subject with the supplied authentication method?"

1039 In response to an authentication query, a responder returns assertions with authentication statements as
1040 follows:

- 1041 • First, rules given in section 3.4.4 for matching against the <Subject> element of the query identify
1042 the assertions that may be returned.
- 1043 • Further, if the <AuthenticationMethod> element is present in the query, at least one
1044 <AuthenticationMethod> element in the set of returned assertions MUST match. It is OPTIONAL
1045 for the complete set of all such matching assertions to be returned in the response.

1046 The <Subject> element in the returned assertions MUST be identical to the <Subject> element of the
1047 query. If the <ConfirmationMethod> element is present in the query, at least one
1048 <ConfirmationMethod> element in the response MUST match. It is OPTIONAL for the complete set of
1049 all such matching assertions to be returned in the response.

1050 The following schema fragment defines the <AuthenticationQuery> type and its
1051 **AuthenticationQueryType** complex type:

```

1052     <element name="AuthenticationQuery"
1053 type="samlp:AuthenticationQueryType" />
1054     <complexType name="AuthenticationQueryType" >
1055         <complexContent>
1056             <extension base="samlp:SubjectQueryAbstractType" >
1057                 <attribute name="AuthenticationMethod"
1058 type="anyURI" />
1059             </extension>
1060         </complexContent>
1061     </complexType>

```

3.3.4 Element <AttributeQuery>

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

Resource [Optional]

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

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

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

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

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

```

1082     <element name="AttributeQuery" type="samlp:AttributeQueryType" />
1083     <complexType name="AttributeQueryType" >
1084         <complexContent>
1085             <extension base="samlp:SubjectQueryAbstractType" >
1086                 <sequence>
1087                     <element ref="saml:AttributeDesignator"
1088                         minOccurs="0"
1089 maxOccurs="unbounded" />
1090                 </sequence>
1091                 <attribute name="Resource" type="anyURI reference"
1092 use="optional" />
1093             </extension>
1094         </complexContent>
1095     </complexType>

```

3.3.5 Element <AuthorizationDecisionQuery>

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

1102 Resource [Required]
 1103 A URI reference indicating the resource for which authorization is requested.
 1104 <Action> [One or More]
 1105 The actions for which authorization is requested.
 1106 <Evidence> [Any Number]
 1107 An assertion that the responder MAY rely on in making its response.
 1108 The following schema fragment defines the <AuthorizationDecisionQuery> element and its
 1109 **AuthorizationDecisionQueryType** complex type:

```

1110     <element name="AuthorizationDecisionQuery"
1111 type="samlp:AuthorizationDecisionQueryType" />
1112     <complexType name="AuthorizationDecisionQueryType">
1113       <complexContent>
1114         <extension base="samlp:SubjectQueryAbstractType">
1115           <sequence>
1116             <element ref="saml:Action"
1117 maxOccurs="unbounded" />
1118             <element ref="saml:Evidence"
1119 minOccurs="0"
1120 maxOccurs="unbounded" />
1121           </sequence>
1122           <attribute name="Resource" type="anyURI"
1123 use="required" />
1124         </extension>
1125       </complexContent>
1126     </complexType>

```

1127 3.4 Responses

1128 The following sections define the SAML constructs that contain response information.

1129 3.4.1 Complex Type ResponseAbstractType

1130 All SAML responses are of types that are derived from the abstract **ResponseAbstractType** complex
 1131 type. This type defines common attributes and elements that are associated with all SAML responses:

1132 ResponseID [Required]
 1133 An identifier for the response. It is of type **IDType**, and MUST follow the requirements specified
 1134 by that type for identifier uniqueness.
 1135 InResponseTo [Optional]
 1136 A reference to the identifier of the request to which the response corresponds, if any. If the
 1137 response is not generated in response to a request, or if the RequestID of a request cannot be
 1138 determined (because the request is malformed), then this attribute MUST NOT be present.
 1139 Otherwise, it MUST be present and match the value of the corresponding RequestID attribute.
 1140 MajorVersion [Required]
 1141 The major version of this response. The identifier for the version of SAML defined in this
 1142 specification is 1. Processing of this attribute is specified in Section 3.4.4.
 1143 MinorVersion [Required]
 1144 The minor version of this response. The identifier for the version of SAML defined in this
 1145 specification is 0. Processing of this attribute is specified in Section 3.4.4.
 1146 IssueInstant [Optional]
 1147 The time instant of issue of the response. The time value is encoded in UTC as described in
 1148 section 1.2.1.

1149 Recipient [Optional]
1150 The intended recipient of this response. This is useful to prevent malicious forwarding of
1151 responses to unintended recipients, a protection that is required by some use profiles. It is set by
1152 the generator of the response to a URI reference that identifies the intended recipient. If present,
1153 the actual recipient MUST check that the URI reference identifies the recipient or a resource
1154 managed by the recipient. If it does not, the response MUST be discarded.

1155 <Signature> [Optional]
1156 An XML Signature that authenticates the assertion, see section 5.

1157 The following schema fragment defines the **ResponseAbstractType** complex type:

```
1158     <complexType name="ResponseAbstractType" abstract="true">  
1159         <sequence>  
1160             <element ref="ds:Signature" minOccurs="0"/>  
1161         </sequence>  
1162         <attribute name="ResponseID" type="saml:IDType" use="required"/>  
1163         <attribute name="InResponseTo" type="saml:IDReferenceType"  
1164             use="optional"/>  
1165         <attribute name="MajorVersion" type="integer" use="required"/>  
1166         <attribute name="MinorVersion" type="integer" use="required"/>  
1167         <attribute name="IssueInstant" type="dateTime" use="required"/>  
1168         <attribute name="Recipient" type="anyURI" use="optional"/>  
1169     </complexType>
```

1170 **3.4.2 Element <Response>**

1171 The <Response> element specifies the status of the corresponding SAML request and a list of zero or
1172 more assertions that answer the request. It has the complex type **ResponseType**, which extends
1173 **ResponseAbstractType** by adding the following elements (in an unbounded mixture):

1174 <Status> [Required] (see Section 3.4.3)
1175 A code representing the status of the corresponding request.

1176 <Assertion> [Any Number] (see Section 2.3.2)
1177 Specifies an assertion by value.

1178 The following schema fragment defines the <Response> element and its **ResponseType** complex type:

```
1179     <element name="Response" type="samlp:ResponseType"/>  
1180     <complexType name="ResponseType">  
1181         <complexContent>  
1182             <extension base="samlp:ResponseAbstractType">  
1183                 <sequence>  
1184                     <element ref="samlp:Status"/>  
1185                     <element ref="saml:Assertion"  
1186                         minOccurs="0" maxOccurs="unbounded"/>  
1187                 </sequence>  
1188             </extension>  
1189         </complexContent>  
1190     </complexType>
```

1191 **3.4.3 Element <Status>**

1192 The <Status> element :

1193 <StatusCode> [Required]
1194 A code representing the status of the corresponding request.

1195 <StatusMessage> [Any Number]
1196 A message which MAY be returned to an operator.

1197 <StatusDetail> [Optional]
1198 Specifies additional information concerning an error condition.
1199 The following schema fragment defines the <Status> element and its **StatusType** complex type:

```
1200 <element name="Status" type="samlp:StatusType" />  
1201 <complexType name="StatusType">  
1202 <sequence>  
1203 <element ref="samlp:StatusCode" />  
1204 <element ref="samlp:StatusMessage"  
1205 minOccurs="0" maxOccurs="unbounded" />  
1206 <element ref="samlp:StatusDetail" minOccurs="0" />  
1207 </sequence>  
1208 </complexType>
```

1209 3.4.3.1 Element <StatusCode>

1210 The <StatusCode> element specifies one or more nested codes representing the status of the
1211 corresponding request. top-most code value MUST be one of the values defined below. Subsequent
1212 nested code values, if present, may provide more specific information concerning a particular error.

1213 Value [Required]

1214 The status code value as defined below.

1215 <StatusCode> [Optional]

1216 An optional subordinate status code value that provides more specific information on an error
1217 condition.

1218 The following top-level **Status Code** Value QNames are defined. The responder MUST NOT include a
1219 code not listed below except by nesting it below one of the listed values.

1220 Success

1221 The request succeeded.

1222 VersionMismatch

1223 The receiver could not process the request because the version was incorrect.

1224 Receiver

1225 The request could not be performed due to an error at the receiving end.

1226 Sender

1227 The request could not be performed due to an error in the sender or in the request

1228 The following second-level status codes are referenced at various places in the specification. Additional
1229 subcodes MAY be defined in future versions of the SAML specification.

1230 RequestVersionTooHigh

1231 The protocol version specified in the request is a major upgrade from the highest protocol version
1232 supported by the responder.

1233 RequestVersionTooLow

1234 The responder cannot respond to the particular request using the SAML version specified in the
1235 request because it is too low.

1236 RequestVersionDeprecated

1237 The responder does not respond to any requests with the protocol version specified in the
1238 request.

1239 TooManyResponses

1240 The response would contain more elements than the responder will return.

1241 RequestDenied

1242 The responder is able to process the request but has chosen not to respond. MAY be used when
1243 the responder is concerned about the security context of the request or the sequence of requests
1244 received from a particular client.

1245 All status code values defined in this document are QNames associated with the SAML protocol
1246 namespace [SAML] and MUST be prefixed appropriately when they appear in SAML messages. SAML
1247 extensions and SAML Responders are free to define more specific status codes in other namespaces,
1248 but MAY NOT define additional codes in either the SAML assertion or protocol namespaces.

1249 The QNames defined as status codes SHOULD only be used in the StatusCode element's Value attribute
1250 and have the above semantics only in that context.

1251 The following schema fragment defines the <StatusCode> element and its **StatusMessageType** complex
1252 type:

```
1253     <element name="StatusCode" type="samlp:StatusCodeType"/>  
1254     <complexType name="StatusCodeType">  
1255         <sequence>  
1256             <element ref="samlp:StatusCode" minOccurs="0"/>  
1257         </sequence>  
1258         <attribute name="Value" type="QName" use="required"/>  
1259     </complexType>
```

1260 3.4.3.2 Element <StatusMessage>

1261 The <StatusMessage> element specifies a message that MAY be returned to an operator:

1262 The following schema fragment defines the <StatusMessage> element and its **StatusMessageType**
1263 complex type:

```
1264     <element name="StatusMessage" type="string"/>
```

1265 3.4.3.3 Element <StatusDetail>

1266 The <StatusDetail> element MAY be used to specify additional information concerning an error
1267 condition.

1268 The following schema fragment defines the <StatusDetail> element and its **StatusDetailType**
1269 complex type:

```
1270     <element name="StatusDetail" type="samlp:StatusDetailType"/>  
1271     <complexType name="StatusDetailType">  
1272         <sequence>  
1273             <any namespace="##any"  
1274                 processContents="lax" minOccurs="0"  
1275                 maxOccurs="unbounded"/>  
1276         </sequence>  
1277     </complexType>
```

1278 3.4.4 Responses to <AuthenticationQuery> and 1279 <AttributeQuery>

1280 Responses to Authentication and Attribute queries are constructed by matching against the
1281 <saml:Subject> element found within the <AuthenticationQuery> or <AttributeQuery>
1282 elements. In response to these queries, every assertion returned by a SAML responder MUST contain at
1283 least one statement whose <saml:Subject> element **strongly matches** the <saml:Subject>
1284 element found in the query.

1285 A <saml:Subject> element S1 strongly matches S2 if and only if:

- 1286 1 If S2 includes a <saml:NameIdentifier> element, then S1 must include an identical
1287 <saml:NameIdentifier> element.

1288 2 If S2 includes a <saml:SubjectConfirmation> element, then S1 must include an identical
1289 <saml:SubjectConfirmation> element.

1290 If the responder cannot provide an assertion with any statement(s) satisfying the constraints expressed
1291 by a query, the <saml:Response> element MUST NOT contain an <Assertion> element and MUST
1292 include a <saml:StatusCode> with value "Success". It MAY return a <saml:StatusMessage> with
1293 additional information.

1294 4 SAML Versioning

1295 SAML version information appears in the following elements:

- 1296 • <Assertion>
- 1297 • <Request>
- 1298 • <Response>

1299 The version numbering of the SAML assertion is independent of the version number of the SAML
1300 request-response protocol. The version information for each consists of a major version number and a
1301 minor version number, both of which are integers. In accordance with industry practice a version number
1302 SHOULD be presented to the user in the form *Major.Minor*. This document defines SAML Assertions 1.0
1303 and SAML Protocol 1.0.

1304 The version number $Major_B.Minor_B$ is higher than the version number $Major_A.Minor_A$ if and only if:

1305 $Major_B > Major_A \vee ((Major_B = Major_A) \wedge Minor_B > Minor_A)$

1306 Each revision of SAML SHALL assign version numbers to assertions, requests, and responses that are
1307 the same as or higher than the corresponding version number in the SAML version that immediately
1308 preceded it.

1309 New versions of SAML SHALL assign new version numbers as follows:

- 1310 • **Documentation change:** $(Major_B = Major_A) \wedge (Minor_B > Minor_A)$
1311 If the major and minor version numbers are unchanged, the new version *B* only introduces
1312 changes to the documentation that raise no compatibility issues with an implementation of version
1313 *A*.
- 1314 • **Minor upgrade:** $(Major_B = Major_A) \wedge (Minor_B > Minor_A)$
1315 If the major version number of versions *A* and *B* are the same and the minor version number of *B*
1316 is higher than that of *A*, the new SAML version MAY introduce changes to the SAML schema and
1317 semantics but any changes that are introduced in *B* SHALL be compatible with version *A*.
- 1318 • **Major upgrade:** $Major_B > Major_A$
1319 If the major version of *B* number is higher than the major version of *A*, Version *B* MAY introduce
1320 changes to the SAML schema and semantics that are incompatible with *A*.

1321 4.1 Assertion Version

1322 A SAML authority MUST NOT issue any assertion whose version number is not supported.

1323 A SAML relying party MUST reject any assertion whose major version number is not supported.

1324 A SAML relying party MAY reject any assertion whose version number is higher than the highest
1325 supported version.

1326 4.2 Request Version

1327 A SAML authority SHOULD issue requests that specify the highest SAML version supported by both the
1328 sender and recipient.

1329 If the SAML authority does not know the capabilities of the recipient it should assume that it supports the
1330 highest SAML version supported by the sender.

1331 4.3 Response Version

1332 A SAML authority MUST NOT issue responses that specify a higher SAML version number than the
1333 corresponding request.

1334 A SAML authority MUST NOT issue a response that has a major version number that is lower than the
1335 major version number of the corresponding request except to report the error
1336 RequestVersionTooHigh.

1337 An error response resulting from incompatible protocol versions MUST result in reporting a top-level
1338 StatusCode value of VersionMismatch, and MAY result in reporting one of the following second-level
1339 values:

1340 RequestVersionTooHigh
1341 The protocol version specified in the request is a major upgrade from the highest protocol version
1342 supported by the responder.

1343 RequestVersionTooLow
1344 The responder cannot respond to the particular request using the SAML version specified in the
1345 request because it is too low.

1346 RequestVersionDeprecated
1347 The responder does not respond to any requests with the protocol version specified in the
1348 request.

5 SAML and XML Signature Syntax and Processing

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

- An Assertion signed by the asserting party (AP). This supports:
 - (1) Message integrity
 - (2) Authentication of the asserting party to a relying party (RP)
 - (3) If the signature is based on the asserting party's public-private key pair, then it also provides for non-repudiation of origin.
- A SAML request or a SAML response message signed by the message originator. This supports :
 - (4) Message integrity
 - (5) Authentication of message origin to a destination
 - (6) If the signature is based on the originator's public-private key pair, then it also provides for non-repudiation of origin.

Note:

- SAML documents may be the subject of signatures from different packaging contexts. **[XMLSig]** provides a framework for signing in XML and is the framework of choice. However, signing may also take place in the context of S/MIME or Java objects that contain SAML documents. One goal is to ensure compatibility with this type of "foreign" digital signing.
- It is useful to characterize situations when a digital signature is NOT required in SAML.

Assertions:

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

Request/Response messages:

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

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

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

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

5.1 Signing Assertions

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

1388 **5.2 Request/Response Signing**

1389 All SAML requests and responses MAY be signed using the XML Signature. This is reflected in the
1390 schema – Sections 3.2 and 3.4.

1391 **5.3 Signature Inheritance**

1392 **5.3.1 Rationale**

1393 SAML assertions may be embedded within request or response messages or other XML messages,
1394 which may be signed. Request or response messages may themselves be contained within other
1395 messages that are based on other XML messaging frameworks (e.g., SOAP) and the composite object
1396 may be the subject of a signature. Another possibility is that SAML assertions or request/response
1397 messages are embedded within a non-XML messaging object (e.g., MIME package) and signed.

1398 In such a case, the SAML sub-message (Assertion, request, response) may be viewed as inheriting a
1399 signature from the "super-signature" over the enclosing object, provided certain constraints are met.

1400 (1) An assertion may be viewed as inheriting a signature from a super signature, if the super
1401 signature applies all the elements within the assertion.

1402 A SAML request or response may be viewed as inheriting a signature from a super signature, if the super
1403 signature applies to all of the elements within the response.

1404 **5.3.2 Rules for SAML Signature Inheritance**

1405 Signature inheritance occurs when SAML message (assertion/request/response) is not signed but is
1406 enclosed within signed SAML such that the signature applies to all of the elements within the message. In
1407 such a case, the SAML message is said to inherit the signature and may be considered equivalent to the
1408 case where it is explicitly signed. The SAML message inherits the "closest enclosing signature".

1409 But if SAML messages need to be passed around by themselves, or embedded in other messages, they
1410 would need to be signed as per section 5.1

1411 **5.4 XML Signature Profile**

1412 The XML Signature [**XMLSig**] specification calls out a general XML syntax for signing data with many
1413 flexibilities and choices. This section details the constraints on these facilities so that SAML processors
1414 do not have to deal with the full generality of XML Signature processing.

1415 **5.4.1 Signing Formats**

1416 XML Signature has three ways of representing signature in a document: enveloping, enveloped, and
1417 detached.

1418 SAML assertions and protocols MUST use the enveloped signatures for signing assertions and protocols.
1419 SAML processors should support use of RSA signing and verification for public key operations.

1420 **5.4.2 Canonicalization Method**

1421 XML Signature REQUIRES the Canonical XML (omits comments) (<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>). SAML implementations SHOULD use Canonical XML with no comments.

1423 **5.4.3 Transforms**

1424 [XMLSig] REQUIRES the enveloped signature transform [http://www.w3.org/2000/09/xmlsig#enveloped-](http://www.w3.org/2000/09/xmlsig#enveloped-signature)
1425 [signature](http://www.w3.org/2000/09/xmlsig#enveloped-signature)

1426 **5.4.4 KeyInfo**

1427 SAML does not restrict or impose any restrictions in this area. Therefore, following [XMLSig],
1428 <ds:KeyInfo> may be absent.

1429 **5.4.5 Binding Between Statements in a Multi-Statement**
1430 **Assertion**

1431 Use of signing does not affect semantics of statements within assertions in any way, as stated in this
1432 document Sections 1 through 4.

1433

6 SAML Extensions

1434 The SAML schemas support extensibility. An example of an application that extends SAML assertions is
1435 the XTAML system for management of embedded trust roots [XTAML]. The following sections explain
1436 how to use the extensibility features in SAML to create extension schemas.

1437 Note that elements in the SAML schemas are not blocked from substitution, so that all SAML elements
1438 MAY serve as the head element of a substitution group. Also, types are not defined as *final*, so that all
1439 SAML types MAY be extended and restricted. The following sections discuss only elements that have
1440 been specifically designed to support extensibility.

1441

6.1 Assertion Schema Extension

1442 The SAML assertion schema is designed to permit separate processing of the assertion package and the
1443 statements it contains, if the extension mechanism is used for either part.

1444 The following elements are intended specifically for use as extension points in an extension schema; their
1445 types are set to *abstract*, so that the use of an *xsi:type* attribute with these elements is REQUIRED:

- 1446 • `<Assertion>`
- 1447 • `<Condition>`
- 1448 • `<Statement>`
- 1449 • `<SubjectStatement>`
- 1450 • `<AdviceElement>`

1451 In addition, the following elements that are directly usable as part of SAML MAY be extended:

- 1452 • `<AuthenticationStatement>`
- 1453 • `<AuthorizationDecisionStatement>`
- 1454 • `<AttributeStatement>`
- 1455 • `<AudienceRestrictionCondition>`

1456 Finally, the following elements are defined to allow elements from arbitrary namespaces within them,
1457 which serves as a built-in extension point without requiring an extension schema:

- 1458 • `<AttributeValue>`
- 1459 • `<Advice>`

1460

6.2 Protocol Schema Extension

1461 The following elements are intended specifically for use as extension points in an extension schema; their
1462 types are set to *abstract*, so that the use of an *xsi:type* attribute with these elements is REQUIRED:

- 1463 • `<Query>`
- 1464 • `<SubjectQuery>`

1465 In addition, the following elements that are directly usable as part of SAML MAY be extended:

- 1466 • `<Request>`
- 1467 • `<AuthenticationQuery>`
- 1468 • `<AuthorizationDecisionQuery>`
- 1469 • `<AttributeQuery>`
- 1470 • `<Response>`

1471 6.3 Use of Type Derivation and Substitution Groups

1472 W3C XML Schema [Schema1] provides two principal mechanisms for specifying an element of an
1473 extended type: type derivation and substitution groups.

1474 For example, a `<Statement>` element can be assigned the type **NewStatementType** by means of the
1475 `xsi:type` attribute. For such an element to be schema-valid, **NewStatementType** needs to be derived
1476 from **StatementType**. The following example of a SAML assertion assumes that the extension schema
1477 (represented by the `new:` prefix) has defined this new type:

```
1478 <saml:Assertion ...>  
1479   <saml:Statement xsi:type="new:NewStatementType">  
1480     ...  
1481   </saml:Statement>  
1482 </saml:Assertion>
```

1483 Alternatively, the extension schema can define a `<NewStatement>` element that is a member of a
1484 substitution group that has `<Statement>` as a head element. For the substituted element to be schema-
1485 valid, it needs to have a type that matches or is derived from the head element's type. The following is an
1486 example of an extension schema fragment that defines this new element:

```
1487 <xsd:element "NewStatement" type="new:NewStatementType"  
1488   substitutionGroup="saml:Statement" />
```

1489 The substitution group declaration allows the `<NewStatement>` element to be used anywhere the SAML
1490 `<Statement>` element can be used. The following is an example of a SAML assertion that uses the
1491 extension element:

```
1492 <saml:Assertion ...>  
1493   <new:NewStatement>  
1494     ...  
1495   </new:NewStatement>  
1496 </saml:Assertion>
```

1497 The choice of extension method has no effect on the semantics of the XML document but does have
1498 implications for interoperability.

1499 The advantages of type derivation are as follows:

- 1500 • A document can be more fully interpreted by a parser that does not have access to the extension
1501 schema because a "native" SAML element is available.
- 1502 • At the time of writing, some W3C XML Schema validators do not support substitution groups,
1503 whereas the `xsi:type` attribute is widely supported.

1504 The advantage of substitution groups is that a document can be explained without the need to explain the
1505 functioning of the `xsi:type` attribute.

1506 7 SAML-Defined Identifiers

1507 The following sections define URI-based identifiers for common authentication protocols and actions.

1508 Where possible an existing URN is used to specify a protocol. In the case of IETF protocols the URN of
1509 the most current RFC that specifies the protocol is used. URI references created specifically for SAML
1510 have the initial stem:

1511 `urn:oasis:names:tc:SAML:1.0:`

1512 7.1 Authentication Method Identifiers

1513 The `<AuthenticationMethod>` and `<SubjectConfirmationMethod>` elements perform different
1514 functions within the SAML architecture, although both can refer to the same underlying mechanisms.
1515 `<AuthenticationMethod>` is a part of an Authentication Statement, which describes an authentication
1516 act which occurred in the past. The `<AuthenticationMethod>` indicates how that authentication was
1517 done. Note that the authentication statement does not provide the means to perform that authentication,
1518 such as a password, key or certificate.

1519 In contrast, `<SubjectConfirmationMethod>` is a part of the `<SubjectConfirmation>`, which is
1520 used to allow the Relying Party to confirm that the request or message came from the System Entity that
1521 corresponds to the Subject in the statement. The `<SubjectConfirmationMethod>` indicates the
1522 method that the Relying Party can use to do this in the future. This may or may not have any relationship
1523 to an authentication that was performed previously. Unlike the Authentication Method, the
1524 `<SubjectConfirmationMethod>` may be accompanied with some piece of information, such as a
1525 certificate or key, which will allow the Relying Party to perform the necessary check.

1526 Subject Confirmation Methods are defined in the SAML Profile or Profiles in which they are used
1527 **[SAMLBind]**. Additional methods may be added by defining new profiles or by private agreement.

1528 The following identifiers refer to SAML specified Authentication methods.

1529 **7.1.1 Password**

1530 **URI:** `urn:oasis:names:tc:SAML:1.0:am:password`

1531 The authentication was performed by means of a password.

1532 **7.1.2 Kerberos**

1533 **URI:** `urn:ietf:rfc:1510`

1534 The authentication was performed by means of the Kerberos protocol **[RFC 1510]**, an instantiation of the
1535 Needham-Schroeder symmetric key authentication mechanism **[Needham78]** .

1536 **7.1.3 Secure Remote Password (SRP)**

1537 **URI:** `urn:ietf:rfc:2945`

1538 The authentication was performed by means of Secure Remote Password
1539 protocol as specified in **[RFC 2945]**

1540 **7.1.4 Hardware Token**

1541 **URI:** urn:oasis:names:tc:SAML:1.0:am:HardwareToken

1542 The authentication was performed by means of an unspecified hardware token.

1543 **7.1.5 SSL/TLS Certificate Based Client Authentication:**

1544 **URI:** urn:ietf:rfc:2246

1545 The authentication was performed using either the SSL or TLS protocol with certificate based client
1546 authentication. TLS is described in **[RFC 2246]**.

1547 **7.1.6 X.509 Public Key**

1548 **URI:** urn:oasis:names:tc:SAML:1.0:am:X509-PKI

1549 The authentication was performed by some (unspecified) mechanism on a key authenticated by means of
1550 an X.509 PKI **[X.500][PKIX]**. It may have been one of the mechanisms for which a more specific identifier
1551 has been defined below.

1552 **7.1.7 PGP Public Key**

1553 **URI:** urn:oasis:names:tc:SAML:1.0:am:PGP

1554 The authentication was performed by some (unspecified) mechanism on a key authenticated by means of
1555 a PGP web of trust **[PGP]**. It may have been one of the mechanisms for which a more specific identifier
1556 has been defined below.

1557 **7.1.8 SPKI Public Key**

1558 **URI:** urn:oasis:names:tc:SAML:1.0:am:SPKI

1559 The authentication was performed by some (unspecified) mechanism on a key authenticated by means of
1560 a SPKI PKI **[SPKI]**. It may have been one of the mechanisms for which a more specific identifier has
1561 been defined below.

1562 **7.1.9 XKMS Public Key**

1563 **URI:** urn:oasis:names:tc:SAML:1.0:am:XKMS

1564 The authentication was performed by some (unspecified) mechanism on a key authenticated by means of
1565 a XKMS trust service **[XKMS]**. It may have been one of the mechanisms for which a more specific
1566 identifier has been defined below.

1567 **7.1.10 XML Digital Signature**

1568 **URI:** urn:ietf:rfc:3075

1569 The authentication was performed by means of an XML digital signature **[RFC 3075]**.

1570 **7.2 Action Namespace Identifiers**

1571 The following identifiers MAY be used in the `Namespace` attribute of the `<Action>` element (see Section
1572 2.4.4.1) to refer to common sets of actions to perform on resources.

1573 **7.2.1 Read/Write/Execute/Delete/Control**

1574 **URI:** urn:oasis:names:tc:SAML:1.0:action:rwedc

1575 Defined actions:

1576 `Read Write Execute Delete Control`

1577 These actions are interpreted in the normal manner, i.e.

1578 `Read`

1579 `The subject may read the resource`

1580 `Write`

1581 `The subject may modify the resource`

1582 `Execute`

1583 `The subject may execute the resource`

1584 `Delete`

1585 `The subject may delete the resource`

1586 `Control`

1587 `The subject may specify the access control policy for the resource`

1588 **7.2.2 Read/Write/Execute/Delete/Control with Negation**

1589 **URI:** urn:oasis:names:tc:SAML:1.0:action:rwedc-negation

1590 Defined actions:

1591 `Read Write Execute Delete Control ~Read ~Write ~Execute ~Delete ~Control`

1592 The actions specified in section 7.2.1 are interpreted in the same manner described there. Actions
1593 prefixed with a tilde (~) are negated permissions and are used to affirmatively specify that the stated
1594 permission is denied. Thus a subject described as being authorized to perform the action `~Read` is
1595 affirmatively denied read permission.

1596 A SAML authority MUST NOT authorize both an action and its negated form.

1597 **7.2.3 Get/Head/Put/Post**

1598 **URI:** urn:oasis:names:tc:SAML:1.0:action:ghpp

1599 Defined actions:

1600 `GET HEAD PUT POST`

1601 These actions bind to the corresponding HTTP operations. For example a subject authorized to perform
1602 the `GET` action on a resource is authorized to retrieve it.

1603 The `GET` and `HEAD` actions loosely correspond to the conventional read permission and the `PUT` and
1604 `POST` actions to the write permission. The correspondence is not exact however since an HTTP `GET`
1605 operation may cause data to be modified and a `POST` operation may cause modification to a resource

1606 other than the one specified in the request. For this reason a separate Action URI reference specifier is
1607 provided.

1608 **7.2.4 UNIX File Permissions**

1609 **URI:** urn:oasis:names:tc:SAML:1.0:action:unix

1610 The defined actions are the set of UNIX file access permissions expressed in the numeric (octal) notation.

1611 The action string is a four digit numeric code:

1612 *extended user group world*

1613 Where the *extended* access permission has the value

1614 +2 if sgid is set

1615 +4 if suid is set

1616 The *user group* and *world* access permissions have the value

1617 +1 if execute permission is granted

1618 +2 if write permission is granted

1619 +4 if read permission is granted

1620 For example, 0754 denotes the UNIX file access permission: user read, write and execute, group read
1621 and execute and world read.

1622 8 SAML Schema Listings

1623 The following sections contain complete listings of the assertion and protocol schemas for SAML.

1624 8.1 Assertion Schema

1625 Following is a complete listing of the SAML assertion schema [SAML-XSD].

```
1626 <?xml version="1.0" encoding="UTF-8"?>
1627 <!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-
1628 Baker (VeriSign Inc.) -->
1629 <schema
1630     targetNamespace="urn:oasis:names:tc:SAML:1.0:assertion"
1631     xmlns="http://www.w3.org/2001/XMLSchema"
1632     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
1633     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1634     elementFormDefault="unqualified">
1635     <import namespace="http://www.w3.org/2000/09/xmldsig#"
1636     schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-
1637     schema.xsd"/>
1638     <annotation>
1639         <documentation>cs-sstc-schema-assertion-00</documentation>
1640     </annotation>
1641     <simpleType name="IDType">
1642         <restriction base="string"/>
1643     </simpleType>
1644     <simpleType name="IDReferenceType">
1645         <restriction base="string"/>
1646     </simpleType>
1647     <simpleType name="DecisionType">
1648         <restriction base="string">
1649             <enumeration value="Permit"/>
1650             <enumeration value="Deny"/>
1651             <enumeration value="Indeterminate"/>
1652         </restriction>
1653     </simpleType>
1654     <element name="AssertionIDReference" type="saml:IDReferenceType"/>
1655     <element name="Assertion" type="saml:AssertionType"/>
1656     <complexType name="AssertionType">
1657         <sequence>
1658             <element ref="saml:Conditions" minOccurs="0"/>
1659             <element ref="saml:Advice" minOccurs="0"/>
1660             <choice maxOccurs="unbounded">
1661                 <element ref="saml:Statement"/>
1662                 <element ref="saml:SubjectStatement"/>
1663                 <element ref="saml:AuthenticationStatement"/>
1664                 <element ref="saml:AuthorizationDecisionStatement"/>
1665                 <element ref="saml:AttributeStatement"/>
1666             </choice>
1667             <element ref="ds:Signature" minOccurs="0"/>
1668         </sequence>
1669         <attribute name="MajorVersion" type="integer" use="required"/>
1670         <attribute name="MinorVersion" type="integer" use="required"/>
1671         <attribute name="AssertionID" type="saml:IDType" use="required"/>
```

```

1672         <attribute name="Issuer" type="string" use="required"/>
1673         <attribute name="IssueInstant" type="dateTime" use="required"/>
1674     </complexType>
1675     <element name="Conditions" type="saml:ConditionsType"/>
1676     <complexType name="ConditionsType">
1677         <choice minOccurs="0" maxOccurs="unbounded">
1678             <element ref="saml:AudienceRestrictionCondition"/>
1679             <element ref="saml:Condition"/>
1680         </choice>
1681         <attribute name="NotBefore" type="dateTime" use="optional"/>
1682         <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>
1683     </complexType>
1684     <element name="Condition" type="saml:ConditionAbstractType"/>
1685     <complexType name="ConditionAbstractType" abstract="true"/>
1686     <element name="AudienceRestrictionCondition"
1687         type="saml:AudienceRestrictionConditionType"/>
1688     <complexType name="AudienceRestrictionConditionType">
1689         <complexContent>
1690             <extension base="saml:ConditionAbstractType">
1691                 <sequence>
1692                     <element ref="saml:Audience"
1693 maxOccurs="unbounded"/>
1694                 </sequence>
1695             </extension>
1696         </complexContent>
1697     </complexType>
1698     <element name="Audience" type="anyURI"/>
1699     <element name="Advice" type="saml:AdviceType"/>
1700     <complexType name="AdviceType">
1701         <choice minOccurs="0" maxOccurs="unbounded">
1702             <element ref="saml:AssertionIDReference"/>
1703             <element ref="saml:Assertion"/>
1704             <any namespace="##other" processContents="lax"/>
1705         </choice>
1706     </complexType>
1707     <element name="Statement" type="saml:StatementAbstractType"/>
1708     <complexType name="StatementAbstractType" abstract="true"/>
1709     <element name="SubjectStatement"
1710 type="saml:SubjectStatementAbstractType"/>
1711     <complexType name="SubjectStatementAbstractType" abstract="true">
1712         <complexContent>
1713             <extension base="saml:StatementAbstractType">
1714                 <sequence>
1715                     <element ref="saml:Subject"/>
1716                 </sequence>
1717             </extension>
1718         </complexContent>
1719     </complexType>
1720     <element name="Subject" type="saml:SubjectType"/>
1721     <complexType name="SubjectType">
1722         <choice>
1723             <sequence>
1724                 <element ref="saml:NameIdentifier"/>
1725                 <element ref="saml:SubjectConfirmation"
1726 minOccurs="0"/>
1727             </sequence>
1728             <element ref="saml:SubjectConfirmation"/>

```

```

1729         </choice>
1730     </complexType>
1731     <element name="NameIdentifier" type="saml:NameIdentifierType"/>
1732     <complexType name="NameIdentifierType">
1733         <simpleContent>
1734             <extension base="string">
1735                 <attribute name="NameQualifier" type="string"
1736 use="optional"/>
1737                 <attribute name="Format" type="anyURI"
1738 use="optional"/>
1739             </extension>
1740         </simpleContent>
1741     </complexType>
1742     <element name="SubjectConfirmation"
1743 type="saml:SubjectConfirmationType"/>
1744     <complexType name="SubjectConfirmationType">
1745         <sequence>
1746             <element ref="saml:ConfirmationMethod"
1747 maxOccurs="unbounded"/>
1748             <element ref="saml:SubjectConfirmationData" minOccurs="0"/>
1749             <element ref="ds:KeyInfo" minOccurs="0"/>
1750         </sequence>
1751     </complexType>
1752     <element name="SubjectConfirmationData" type="anyType"/>
1753     <element name="ConfirmationMethod" type="anyURI"/>
1754     <element name="AuthenticationStatement"
1755         type="saml:AuthenticationStatementType"/>
1756     <complexType name="AuthenticationStatementType">
1757         <complexContent>
1758             <extension base="saml:SubjectStatementAbstractType">
1759                 <sequence>
1760                     <element ref="saml:SubjectLocality"
1761 minOccurs="0"/>
1762                     <element ref="saml:AuthorityBinding"
1763 minOccurs="0" maxOccurs="unbounded"/>
1764                 </sequence>
1765                 <attribute name="AuthenticationMethod"
1766 type="anyURI"/>
1767                 <attribute name="AuthenticationInstant"
1768 type="dateTime"/>
1769             </extension>
1770         </complexContent>
1771     </complexType>
1772     <element name="SubjectLocality"
1773         type="saml:SubjectLocalityType"/>
1774     <complexType name="SubjectLocalityType">
1775         <attribute name="IPAddress" type="string" use="optional"/>
1776         <attribute name="DNSAddress" type="string" use="optional"/>
1777     </complexType>
1778     <element name="AuthorityBinding" type="saml:AuthorityBindingType"/>
1779     <complexType name="AuthorityBindingType">
1780         <attribute name="AuthorityKind" type="QName" use="required"/>
1781         <attribute name="Location" type="anyURI" use="required"/>
1782         <attribute name="Binding" type="anyURI" use="required"/>
1783     </complexType>
1784     <element name="AuthorizationDecisionStatement"
1785 type="saml:AuthorizationDecisionStatementType"/>

```

```

1786     <complexType name="AuthorizationDecisionStatementType">
1787         <complexContent>
1788             <extension base="saml:SubjectStatementAbstractType">
1789                 <sequence>
1790                     <element ref="saml:Action"
1791 maxOccurs="unbounded"/>
1792                     <element ref="saml:Evidence" minOccurs="0"/>
1793                 </sequence>
1794                 <attribute name="Resource" type="anyURI"
1795 use="required"/>
1796                 <attribute name="Decision" type="saml:DecisionType"
1797 use="required"/>
1798             </extension>
1799         </complexContent>
1800     </complexType>
1801     <element name="Action" type="saml:ActionType"/>
1802     <complexType name="ActionType">
1803         <simpleContent>
1804             <extension base="string">
1805                 <attribute name="Namespace" type="anyURI"/>
1806             </extension>
1807         </simpleContent>
1808     </complexType>
1809     <element name="Evidence" type="saml:EvidenceType"/>
1810     <complexType name="EvidenceType">
1811         <choice maxOccurs="unbounded">
1812             <element ref="saml:AssertionIDReference"/>
1813             <element ref="saml:Assertion"/>
1814         </choice>
1815     </complexType>
1816     <element name="AttributeStatement" type="saml:AttributeStatementType"/>
1817     <complexType name="AttributeStatementType">
1818         <complexContent>
1819             <extension base="saml:SubjectStatementAbstractType">
1820                 <sequence>
1821                     <element ref="saml:Attribute"
1822 maxOccurs="unbounded"/>
1823                 </sequence>
1824             </extension>
1825         </complexContent>
1826     </complexType>
1827     <element name="AttributeDesignator"
1828 type="saml:AttributeDesignatorType"/>
1829     <complexType name="AttributeDesignatorType">
1830         <attribute name="AttributeName" type="string" use="required"/>
1831         <attribute name="AttributeNamespace" type="anyURI"
1832 use="required"/>
1833     </complexType>
1834     <element name="Attribute" type="saml:AttributeType"/>
1835     <complexType name="AttributeType">
1836         <complexContent>
1837             <extension base="saml:AttributeDesignatorType">
1838                 <sequence>
1839                     <element ref="saml:AttributeValue"
1840 maxOccurs="unbounded"/>
1841                 </sequence>
1842             </extension>

```

```

1843         </complexContent>
1844     </complexType>
1845     <element name="AttributeValue" type="saml:anyType"/>
1846 </schema>

```

8.2 Protocol Schema

1847 Following is a complete listing of the SAML protocol schema [SAML-P-XSD].

```

1849 <?xml version="1.0" encoding="UTF-8"?>
1850 <!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-
1851 Baker (VeriSign Inc.) -->
1852 <schema
1853     targetNamespace="urn:oasis:names:tc:SAML:1.0:protocol"
1854     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1855     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
1856     xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
1857     xmlns="http://www.w3.org/2001/XMLSchema"
1858     elementFormDefault="unqualified">
1859     <import
1860         namespace="urn:oasis:names:tc:SAML:1.0:assertion"
1861         schemaLocation="cs-sstc-schema-assertion-00.xsd"/>
1862     <import namespace="http://www.w3.org/2000/09/xmldsig#"
1863         schemaLocation="xmldsig-core-schema.xsd"/>
1864     <annotation>
1865         <documentation>cs-sstc-schema-protocol-00.xsd</documentation>
1866     </annotation>
1867     <complexType name="RequestAbstractType" abstract="true">
1868         <sequence>
1869             <element ref="samlp:RespondWith"
1870                 minOccurs="0" maxOccurs="unbounded"/>
1871             <element ref="ds:Signature" minOccurs="0"/>
1872         </sequence>
1873         <attribute name="RequestID" type="saml:IDType" use="required"/>
1874         <attribute name="MajorVersion" type="integer" use="required"/>
1875         <attribute name="MinorVersion" type="integer" use="required"/>
1876         <attribute name="IssueInstant" type="dateTime" use="required"/>
1877     </complexType>
1878     <element name="RespondWith" type="QName"/>
1879     <element name="Request" type="samlp:RequestType"/>
1880     <complexType name="RequestType">
1881         <complexContent>
1882             <extension base="samlp:RequestAbstractType">
1883                 <choice>
1884                     <element ref="samlp:Query"/>
1885                     <element ref="samlp:SubjectQuery"/>
1886                     <element ref="samlp:AuthenticationQuery"/>
1887                     <element
1888 ref="samlp:AuthorizationDecisionQuery"/>
1889                     <element ref="saml:AssertionID"
1890 maxOccurs="unbounded"/>
1891                     <element ref="samlp:AssertionArtifact"
1892 maxOccurs="unbounded"/>
1893                 </choice>
1894             </extension>
1895         </complexContent>

```



```

1897 </complexType>
1898 <element name="AssertionArtifact" type="string"/>
1899 <element name="Query" type="samlp:QueryAbstractType"/>
1900 <complexType name="QueryAbstractType" abstract="true"/>
1901 <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType"/>
1902 <complexType name="SubjectQueryAbstractType" abstract="true">
1903 <complexContent>
1904 <extension base="samlp:QueryAbstractType">
1905 <sequence>
1906 <element ref="saml:Subject"/>
1907 </sequence>
1908 </extension>
1909 </complexContent>
1910 </complexType>
1911 <element name="AuthenticationQuery"
1912 type="samlp:AuthenticationQueryType"/>
1913 <complexType name="AuthenticationQueryType">
1914 <complexContent>
1915 <extension base="samlp:SubjectQueryAbstractType">
1916 <attribute name="AuthenticationMethod"
1917 type="anyURI"/>
1918 </extension>
1919 </complexContent>
1920 </complexType>
1921 <element name="AttributeQuery" type="samlp:AttributeQueryType"/>
1922 <complexType name="AttributeQueryType">
1923 <complexContent>
1924 <extension base="samlp:SubjectQueryAbstractType">
1925 <sequence>
1926 <element ref="saml:AttributeDesignator"
1927 minOccurs="0" maxOccurs="unbounded"/>
1928 </sequence>
1929 <attribute name="Resource" type="anyURI"
1930 use="optional"/>
1931 </extension>
1932 </complexContent>
1933 </complexType>
1934 <element name="AuthorizationDecisionQuery"
1935 type="samlp:AuthorizationDecisionQueryType"/>
1936 <complexType name="AuthorizationDecisionQueryType">
1937 <complexContent>
1938 <extension base="samlp:SubjectQueryAbstractType">
1939 <sequence>
1940 <element ref="saml:Action"
1941 maxOccurs="unbounded"/>
1942 <element ref="saml:Evidence"
1943 minOccurs="0" maxOccurs="unbounded"/>
1944 </sequence>
1945 <attribute name="Resource" type="anyURI"
1946 use="required"/>
1947 </extension>
1948 </complexContent>
1949 </complexType>
1950 <complexType name="ResponseAbstractType" abstract="true">
1951 <sequence>
1952 <element ref="ds:Signature" minOccurs="0"/>
1953 </sequence>

```

```

1954     <attribute name="ResponseID" type="saml:IDType" use="required"/>
1955     <attribute name="InResponseTo" type="saml:IDReferenceType"
1956         use="optional"/>
1957     <attribute name="MajorVersion" type="integer" use="required"/>
1958     <attribute name="MinorVersion" type="integer" use="required"/>
1959     <attribute name="IssueInstant" type="dateTime" use="required"/>
1960     <attribute name="Recipient" type="anyURI" use="optional"/>
1961 </complexType>
1962 <element name="Response" type="samlp:ResponseType"/>
1963 <complexType name="ResponseType">
1964     <complexContent>
1965         <extension base="samlp:ResponseAbstractType">
1966             <sequence>
1967                 <element ref="samlp:Status"/>
1968                 <element ref="saml:Assertion"
1969                     minOccurs="0" maxOccurs="unbounded"/>
1970             </sequence>
1971         </extension>
1972     </complexContent>
1973 </complexType>
1974 <element name="Status" type="samlp:StatusType"/>
1975 <complexType name="StatusType">
1976     <sequence>
1977         <element ref="samlp:StatusCode"/>
1978         <element ref="samlp:StatusMessage"
1979             minOccurs="0" maxOccurs="unbounded"/>
1980         <element ref="samlp:StatusDetail" minOccurs="0"/>
1981     </sequence>
1982 </complexType>
1983 <element name="StatusCode" type="samlp:StatusCodeType"/>
1984 <complexType name="StatusCodeType">
1985     <sequence>
1986         <element ref="samlp:StatusCode" minOccurs="0"/>
1987     </sequence>
1988     <attribute name="Value" type="QName" use="required"/>
1989 </complexType>
1990 <element name="StatusMessage" type="string"/>
1991 <element name="StatusDetail" type="samlp:StatusDetailType"/>
1992 <complexType name="StatusDetailType">
1993     <sequence>
1994         <any namespace="##any"
1995             processContents="lax" minOccurs="0"
1996             maxOccurs="unbounded"/>
1997     </sequence>
1998 </complexType>
1999 </schema>
2000

```

9 References

- 2001
- 2002 [Kern-84] B. Kernighan, Rob Pike *The UNIX Programming Environment*, (March 1984)
2003 Prentice Hall Computer Books;
- 2004 [Needham78] R. Needham et al., *Using Encryption for Authentication in Large Networks of*
2005 *Computers*, Communications of the ACM, Vol. 21 (12), pp. 993-999, December
2006 1978.
- 2007 [PGP] Atkins, D., Stallings, W. and P. Zimmermann, *PGP Message Exchange Formats*,
2008 RFC 1991, August 1996.
- 2009 [PKCS1] B. Kaliski, *PKCS #1: RSA Encryption Version 2.0*, RSA Laboratories, also IETF
2010 RFC 2437, October 1998. <http://www.ietf.org/rfc/rfc2437.txt>
- 2011 [PKCS7] B. Kaliski., *PKCS #7: Cryptographic Message Syntax, Version 1.5*, RFC 2315,
2012 March 1998.
- 2013 [PKIX] R. Housley, W. Ford, W. Polk, D. Solo. *Internet X.509 Public Key Infrastructure*
2014 *Certificate and CRL Profile*. RFC 2459, January 1999.
- 2015 [RFC 1510] J. Kohl, C. Neuman. *The Kerberos Network Authentication Requestor (V5)*.
2016 September 1993. <http://www.ietf.org/rfc/rfc1510.txt>
- 2017 [RFC 2104] H. Krawczyk et al., *HMAC: Keyed Hashing for Message Authentication*,
2018 <http://www.ietf.org/rfc/rfc2104.txt>, IETF RFC 2104, February 1997.
- 2019 [RFC 2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
2020 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997
- 2021 [RFC 2246] T. Dierks, C. Allen. *The TLS Protocol Version 1.0*. January 1999.
2022 <http://www.ietf.org/rfc/rfc2246.txt>
- 2023 [RFC 2396] T. Berners-Lee et. al., *Uniform Resource Identifiers (URI): Generic Syntax*
2024 <http://www.ietf.org/rfc/rfc2396.txt> IETF?
- 2025 [RFC 2630] R. Housley. *Cryptographic Message Syntax*. June 1999.
2026 <http://www.ietf.org/rfc/rfc630.txt>
- 2027 [RFC 2648] R. Moats. *A URN Namespace for IETF Documents*. August 1999.
2028 <http://www.ietf.org/rfc/rfc2648.txt>
- 2029 [RFC 2945] T. Wu, *The SRP Authentication and Key Exchange System*, September 2000,
2030 <http://www.ietf.org/rfc/rfc2945.txt> [RFC 3075] D. Eastlake, J. Reagle, D. Solo.
2031 *XML-Signature Syntax and Processing*. March 2001.
2032 <http://www.ietf.org/rfc/rfc3075.txt>
- 2033 [SAMLBind] P. Mishra et al., *Bindings and Profiles for the OASIS Security Assertion Markup*
2034 *Language (SAML)*, [http://www.oasis-open.org/committees/security/docs/cs-sstc-](http://www.oasis-open.org/committees/security/docs/cs-sstc-bindings-00.pdf)
2035 [bindings-00.pdf](http://www.oasis-open.org/committees/security/docs/cs-sstc-bindings-00.pdf), OASIS, April 2002.
- 2036 [SAMLConform] *Conformance Program Specification for the OASIS Security Assertion Markup*
2037 *Language (SAML)* [http://www.oasis-open.org/committees/security/docs/cs-sstc-](http://www.oasis-open.org/committees/security/docs/cs-sstc-conform-00.pdf)
2038 [conform-00.pdf](http://www.oasis-open.org/committees/security/docs/cs-sstc-conform-00.pdf), OASIS, April 2002.
- 2039 [SAMLGloss] J. Hodges et al., *Glossary for the OASIS Security Assertion Markup Language*
2040 *(SAML)*, [http://www.oasis-open.org/committees/security/docs/cs-sstc-glossary-](http://www.oasis-open.org/committees/security/docs/cs-sstc-glossary-00.pdf)
2041 [00.pdf](http://www.oasis-open.org/committees/security/docs/cs-sstc-glossary-00.pdf), OASIS, April 2002.
- 2042 [SAMLXSD] P. Hallam-Baker et al., *SAML protocol schema*, [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-protocol-00.xsd)
2043 [open.org/committees/security/docs/cs-sstc-schema-protocol-00.xsd](http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-protocol-00.xsd), OASIS,
2044 April 2002.
- 2045 [SAMLSecure] *Security and Privacy Considerations for the OASIS Security Assertion Markup*
2046 *Language (SAML)*, [http://www.oasis-open.org/committees/security/docs/cs-sstc-](http://www.oasis-open.org/committees/security/docs/cs-sstc-sec-consider-00.pdf)
2047 [sec-consider-00.pdf](http://www.oasis-open.org/committees/security/docs/cs-sstc-sec-consider-00.pdf), OASIS, April 2002
- 2048 [SAMLXSD] P. Hallam-Baker et al., *SAML assertion schema*, [http://www.oasis-](http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-assertion-00.xsd)
2049 [open.org/committees/security/docs/cs-sstc-schema-assertion-00.xsd](http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-assertion-00.xsd), OASIS,
2050 April 2002.

2051	[Schema1]	H. S. Thompson et al., <i>XML Schema Part 1: Structures</i> , http://www.w3.org/TR/xmlschema-1/ , World Wide Web Consortium Recommendation, May 2001.
2052		
2053		
2054	[Schema2]	P. V. Biron et al., <i>XML Schema Part 2: Datatypes</i> , http://www.w3.org/TR/xmlschema-2/ , World Wide Web Consortium Recommendation, May 2001.
2055		
2056		
2057	[SPKI]	C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen. <i>SPKI Certificate Theory</i> , RFC 2693, September 1999.
2058		
2059	[UNICODE-C]	M. Davis, M. J. Dürst, http://www.unicode.org/unicode/reports/tr15/tr15-21.html , UNICODE Consortium
2060		
2061	[W3C-CHAR]	M. J. Dürst, <i>Requirements for String Identity Matching and String Indexing</i> http://www.w3.org/TR/WD-charreg , World Wide Web Consortium.
2062		
2063	[W3C-CharMod]	M. J. Dürst,, <i>Unicode Normalization Forms</i> http://www.w3.org/TR/charmod/ , World Wide Web Consortium.
2064		
2065	[X.500]	ITU-T Recommendation X.501: <i>Information Technology - Open Systems Interconnection - The Directory: Models</i> , 1993.
2066		
2067	[XKMS]	W. Ford, P. Hallam-Baker, B. Fox, B. Dillaway, B. LaMacchia, J. Epstein, J. Lapp, XML Key Management Specification (XKMS), W3C Note 30 March 2001, http://www.w3.org/TR/xkms/ .
2068		
2069		
2070	[XML]	T. Bray et. al. <i>Extensible Markup Language (XML) 1.0 (Second Edition)</i> , http://www.w3.org/TR/REC-xml , World Wide Web Consortium.
2071		
2072	[XMLEnc]	<i>XML Encryption Specification</i> , In development.
2073	[XMLSig]	D. Eastlake et al., <i>XML-Signature Syntax and Processing</i> , http://www.w3.org/TR/xmlsig-core/ , World Wide Web Consortium.
2074		
2075	[XMLSig-XSD]	XML Signature Schema available from http://www.w3.org/TR/2000/CR-xmlsig-core-20001031/xmlsig-core-schema.xsd .
2076		
2077	[XTAML]	P. Hallam-Baker, <i>XML Trust Axiom Markup Language 1.0</i> , http://www.xmltrustcenter.org/ , VeriSign Inc. September 2001.
2078		

2079 **Appendix A. Acknowledgments**

2080 The editors would like to acknowledge the contributions of the OASIS SAML Technical Committee, whose
2081 voting members at the time of publication were:

- 2082 • Allen Rogers, Authentica
- 2083 • Irving Reid, Baltimore Technologies
- 2084 • Krishna Sankar, Cisco Systems
- 2085 • Simon Godik, Crosslogix
- 2086 • Gilbert Pilz, E2open
- 2087 • Hal Lockhart, Entegrity
- 2088 • Carlisle Adams, Entrust
- 2089 • Don Flinn, Hitachi
- 2090 • Joe Pato, Hewlett-Packard (co-chair)
- 2091 • Jason Rouault, Hewlett-Packard
- 2092 • Marc Chanliau, Netegrity
- 2093 • Chris McLaren, Netegrity
- 2094 • Prateek Mishra, Netegrity
- 2095 • Charles Knouse, Oblix
- 2096 • Steve Anderson, OpenNetwork
- 2097 • Rob Philpott, RSA Security
- 2098 • Jahan Moreh, Sigaba
- 2099 • Bhavna Bhatnagar, Sun Microsystems
- 2100 • Jeff Hodges, Sun Microsystems (co-chair)
- 2101 • Eve Maler, Sun Microsystems (former chair)
- 2102 • Aravindan Ranganathan, Sun Microsystems
- 2103 • Emily Xu, Sun Microsystems
- 2104 • Bob Morgan, University of Washington
- 2105 • Phillip Hallam-Baker, VeriSign

2106

Appendix B. Notices

2107 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that
2108 might be claimed to pertain to the implementation or use of the technology described in this document or
2109 the extent to which any license under such rights might or might not be available; neither does it
2110 represent that it has made any effort to identify any such rights. Information on OASIS's procedures with
2111 respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights
2112 made available for publication and any assurances of licenses to be made available, or the result of an
2113 attempt made to obtain a general license or permission for the use of such proprietary rights by
2114 implementors or users of this specification, can be obtained from the OASIS Executive Director.

2115 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications,
2116 or other proprietary rights which may cover technology that may be required to implement this
2117 specification. Please address the information to the OASIS Executive Director.

2118 Copyright © The Organization for the Advancement of Structured Information Standards [OASIS] 2001.
2119 All Rights Reserved.

2120 This document and translations of it may be copied and furnished to others, and derivative works that
2121 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published
2122 and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice
2123 and this paragraph are included on all such copies and derivative works. However, this document itself
2124 may not be modified in any way, such as by removing the copyright notice or references to OASIS,
2125 except as needed for the purpose of developing OASIS specifications, in which case the procedures for
2126 copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to
2127 translate it into languages other than English.

2128 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors
2129 or assigns.

2130 This document and the information contained herein is provided on an "AS IS" basis and OASIS
2131 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
2132 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
2133 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.