



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

Document identifier: draft-sstc-core-243

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

Publication date: January 9th 2002

Maturity Level: Committee Last Call Draft **Status:** Interim draft.

Send comments to: security-services-comment@lists.oasis-open.org ~~the editors.~~

Editors:

Phillip Hallam-Baker, VeriSign, ~~(pbaker@verisign.com)~~
Eve Maler, Sun Microsystems, ~~(eve.maler@sun.com)~~
Krishna Sankar, Cisco Systems Inc. ~~(ksankar@cisco.com)~~

Contributors:

Carlisle Adams, Entrust
Scott Cantor, ~~The Ohio State University~~ ~~osu.edu~~
Marc Chanliau, Netegrity
Nigel Edwards, Hewlett-Packard
Marlena Erdos, Tivoli
Stephen Farrell, Baltimore Technologies
Simon Godik, Crosslogic
Jeff Hodges, Oblix
Charles Knouse, Oblix
Chris McLaren, Netegrity
Prateek Mishra, Netegrity
RL "Bob" Morgan, University of Washington
Tim Moses, Entrust
David Orchard, BEA
Joe Pato, Hewlett Packard
Darren Platt, RSA
Irving Reid, Baltimore

32

33

34

**ASSERTIONS AND PROTOCOL FOR THE OASIS SECURITY ASSERTION MARKUP
LANGUAGE (SAML)** **1**

35

1. INTRODUCTION **6**

36

1.1. NOTATION **6**

37

1.2. SCHEMA ORGANIZATION AND NAMESPACES **6**

38

1.3. SAML CONCEPTS (NON-NORMATIVE) **7**

39

1.3.1. Overview **7**

40

1.3.2. SAML and URI-Based Identifiers **8**

41

1.3.3. SAML and Extensibility **9**

42

2. SAML ASSERTIONS **10**

43

2.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS **10**

44

2.2. SIMPLE TYPES **10**

45

2.2.1. Simple Type IDType **10**

46

2.2.2. Simple Type DecisionType **11**

47

2.3. ASSERTIONS **11**

48

2.3.1. Element <AssertionSpecifier> **11**

49

2.3.2. Element <AssertionID> **11**

50

2.3.3. Element <Assertion> **12**

51

2.3.3.1. Element <Conditions> **13**

52

2.3.3.1.1. Attributes NotBefore and NotOnOrAfter **14**

53

2.3.3.1.2. Element <Condition> **14**

54

2.3.3.1.3. Elements <AudienceRestrictionCondition> and <Audience> **14**

55

2.3.3.1.4. Condition Type TargetRestrictionType **15**

56

2.3.3.2. Elements <Advice> and <AdviceElement> **15**

57

2.4. STATEMENTS **16**

58

2.4.1. Element <Statement> **16**

59

2.4.2. Element <SubjectStatement> **16**

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

91	<u>3.4.4. Simple Type StatusCodeType</u>	<u>30</u>
92	4. SAML VERSIONING	31
93	<u>4.1. ASSERTION VERSION</u>	<u>31</u>
94	<u>4.2. REQUEST VERSION</u>	<u>31</u>
95	<u>4.3. RESPONSE VERSION</u>	<u>32</u>
96	5. SAML & XML-SIGNATURE SYNTAX AND PROCESSING	33
97	<u>5.1. SIGNING ASSERTIONS</u>	<u>33</u>
98	<u>5.2. REQUEST /RESPONSE SIGNING</u>	<u>34</u>
99	<u>5.3. SIGNATURE INHERITANCE (A.K.A. SUPER-SIGNATURES & SUB-MESSAGES)</u>	<u>34</u>
100	<u>5.3.1. Rationale</u>	<u>34</u>
101	<u>5.3.2. Rules for SAML Signature Inheritance</u>	<u>34</u>
102	<u>5.4. XML SIGNATURE PROFILE</u>	<u>34</u>
103	<u>5.4.1. Signing formats</u>	<u>34</u>
104	<u>5.4.2. CanonicalizationMethod</u>	<u>34</u>
105	<u>5.4.3. Transforms</u>	<u>35</u>
106	<u>5.4.4. KeyInfo</u>	<u>35</u>
107	<u>5.4.5. Binding between statements in a multi-statement assertion</u>	<u>35</u>
108	<u>5.4.6. Security considerations</u>	<u>35</u>
109	<u>5.4.6.1. Replay Attack</u>	<u>35</u>
110	6. SAML EXTENSIONS	36
111	<u>6.1. ASSERTION SCHEMA EXTENSION</u>	<u>36</u>
112	<u>6.2. PROTOCOL SCHEMA EXTENSION</u>	<u>36</u>
113	<u>6.3. USE OF TYPE DERIVATION AND SUBSTITUTION GROUPS</u>	<u>37</u>
114	7. SAML-DEFINED IDENTIFIERS	38

115	7.1. CONFIRMATION METHOD IDENTIFIERS	38
116	<i>7.1.1. SAML Artifact:</i>	<i>38</i>
117	<i>7.1.2. SAML Artifact (SHA-1):</i>	<i>38</i>
118	<i>7.1.3. Holder of Key:</i>	<i>38</i>
119	<i>7.1.4. Sender Vouches:</i>	<i>38</i>
120	<i>7.1.5. Password (Pass-Through):</i>	<i>38</i>
121	<i>7.1.6. Password (One-Way-Function SHA-1):</i>	<i>39</i>
122	<i>7.1.7. Kerberos</i>	<i>39</i>
123	<i>7.1.8. SSL/TLS Certificate Based Client Authentication:</i>	<i>39</i>
124	<i>7.1.9. Object Authenticator (SHA-1):</i>	<i>39</i>
125	<i>7.1.10. PKCS#7</i>	<i>39</i>
126	<i>7.1.11. Cryptographic Message Syntax</i>	<i>40</i>
127	<i>7.1.12. XML Digital Signature</i>	<i>40</i>
128	7.2. ACTION NAMESPACE IDENTIFIERS	40
129	<i>7.2.1. Read/Write/Execute/Delete/Control:</i>	<i>40</i>
130	<i>7.2.2. Read/Write/Execute/Delete/Control with Negation:</i>	<i>40</i>
131	<i>7.2.3. Get/Head/Put/Post:</i>	<i>41</i>
132	<i>7.2.4. UNIX File Permissions:</i>	<i>41</i>
133	8. SAML SCHEMA LISTINGS	42
134	8.1. ASSERTION SCHEMA	42
135	8.2. PROTOCOL SCHEMA	45
136	9. REFERENCES	49
137	APPENDIX A. NOTICES	51
138	ASSERTIONS AND PROTOCOL FOR THE OASIS SECURITY ASSERTION MARKUP	
139	LANGUAGE (SAML)	1
140	1. INTRODUCTION	6

141	1.1. NOTATION	6
142	1.2. SCHEMA ORGANIZATION AND NAMESPACES	6
143	1.3. SAML CONCEPTS (NON-NORMATIVE)	7
144	2. SAML ASSERTIONS	8
145	2.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS	8
146	2.2. SIMPLE TYPES	8
147	2.2.1. Simple Type IDType	8
148	2.2.2. Simple Type DecisionType	9
149	2.3. ASSERTIONS	9
150	2.3.1. Element <AssertionSpecifier>	9
151	2.3.2. Element <AssertionID>	9
152	2.3.3. Element <Assertion>	10
153	2.3.3.1. Element <Conditions>	11
154	2.3.3.1.1. Attributes NotBefore and NotOnOrAfter	12
155	2.3.3.1.2. Element <Condition>	12
156	2.3.3.1.3. Elements <AudienceRestrictionCondition> and <Audience>	12
157	2.3.3.1.4. Condition Type TargetRestrictionType	13
158	2.3.3.2. Elements <Advice> and <AdviceElement>	13
159	2.4. STATEMENTS	14
160	2.4.1. Element <Statement>	14
161	2.4.2. Element <SubjectStatement>	14
162	2.4.2.1. Element <Subject>	14
163	2.4.2.2. Element <NameIdentifier>	15
164	2.4.2.3. Elements <SubjectConfirmation>, <ConfirmationMethod>, and <SubjectConfirmationData>	15
165	2.4.3. Element <AuthenticationStatement>	16
166	2.4.3.1. Element <AuthenticationLocality>	16
167	2.4.4. Element <AuthorizationDecisionStatement>	17
168	2.4.4.1. Elements <Actions> and <Action>	17
169	2.4.4.2. Element <Evidence>	18
170	2.4.5. Element <AttributeStatement>	18

171	2.4.5.1. Elements \langleAttributeDesignator\rangle and \langleAttribute\rangle	18
172	2.4.5.1.1. Element \langleAttributeValue\rangle	19
173	3. SAML PROTOCOL	20
174	3.1. SCHEMA HEADER AND NAMESPACE DECLARATIONS	20
175	3.2. SIMPLE TYPES	20
176	3.2.1. Simple Type <i>StatusCodeType</i>	20
177	3.3. REQUESTS	21
178	3.3.1. Complex Type <i>RequestAbstractType</i>	21
179	3.3.1.1. Element \langleRespondWith\rangle	21
180	3.3.2. Element \langleRequest\rangle	22
181	3.4. QUERIES	23
182	3.4.1. Element \langleQuery\rangle	23
183	3.4.2. Element \langleSubjectQuery\rangle	23
184	3.4.3. Element \langleAuthenticationQuery\rangle	23
185	3.4.4. Element \langleAttributeQuery\rangle	24
186	3.4.5. Element \langleAuthorizationDecisionQuery\rangle	24
187	3.5. RESPONSES	25
188	3.5.1. Complex Type <i>ResponseAbstractType</i>	25
189	3.5.2. Element \langleResponse\rangle	25
190	3.5.2.1. Element \langleStatusReason\rangle	26
191	4. SAML VERSIONING	27
192	4.1. ASSERTION VERSION	27
193	4.2. REQUEST VERSION	27
194	4.3. RESPONSE VERSION	28
195	5. SAML EXTENSIONS	32
196	5.1. ASSERTION SCHEMA EXTENSION	32

197	5.2. PROTOCOL SCHEMA EXTENSION	32
198	5.3. USE OF TYPE DERIVATION AND SUBSTITUTION GROUPS	33
199	6. SAML-DEFINED IDENTIFIERS	34
200	6.1. CONFIRMATION METHOD IDENTIFIERS	34
201	6.1.1. SAML Artifact:	34
202	6.1.2. SAML Artifact (SHA-1):	34
203	6.1.3. Holder of Key:	34
204	6.1.4. Sender Vouches:	34
205	6.1.5. Password (Pass Through):	34
206	6.1.6. Password (One-Way Function SHA-1):	35
207	6.1.7. Kerberos [Kerberos]	35
208	6.1.8. SSL/TLS Certificate Based Client Authentication:	35
209	6.1.9. Object Authenticator (SHA-1):	35
210	6.1.10. PKCS#7	35
211	6.1.11. Cryptographic Message Syntax	36
212	6.1.12. XML Digital Signature	36
213	6.2. ACTION NAMESPACE IDENTIFIERS	36
214	6.2.1. Read/Write/Execute/Delete/Control:	36
215	6.2.2. Read/Write/Execute/Delete/Control with Negation:	36
216	6.2.3. Get/Head/Put/Post:	37
217	6.2.4. UNIX File Permissions:	37
218	7. SAML SCHEMA LISTINGS	38
219	7.1. ASSERTION SCHEMA	38
220	7.2. PROTOCOL SCHEMA	41
221	8. REFERENCES	44
222	APPENDIX A. NOTICES	46

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 **[RFC2119]**:

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

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

Listings of SAML schemas appear like this.

Example code listings appear like this.

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

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

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

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

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

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

1.2. Schema Organization and Namespaces

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

<http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-22.xsd>

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

<http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-22.xsd>

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

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

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

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

272 **1.3. SAML Concepts (Non-Normative)**

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

276 **1.3.1. Overview**

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

282 Assertions can convey information about authentication acts performed by subjects, attributes of
283 subjects, and authorization decisions about whether subjects are allowed to access certain
284 resources. Assertions are represented as XML constructs and have a nested structure, whereby a
285 single assertion might contain several different internal statements about authentication,
286 authorization, and attributes. Note that authentication assertions merely describe acts of
287 authentication that happened previously; checking and revoking of credentials is outside the scope
288 of this version of SAML.

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

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

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

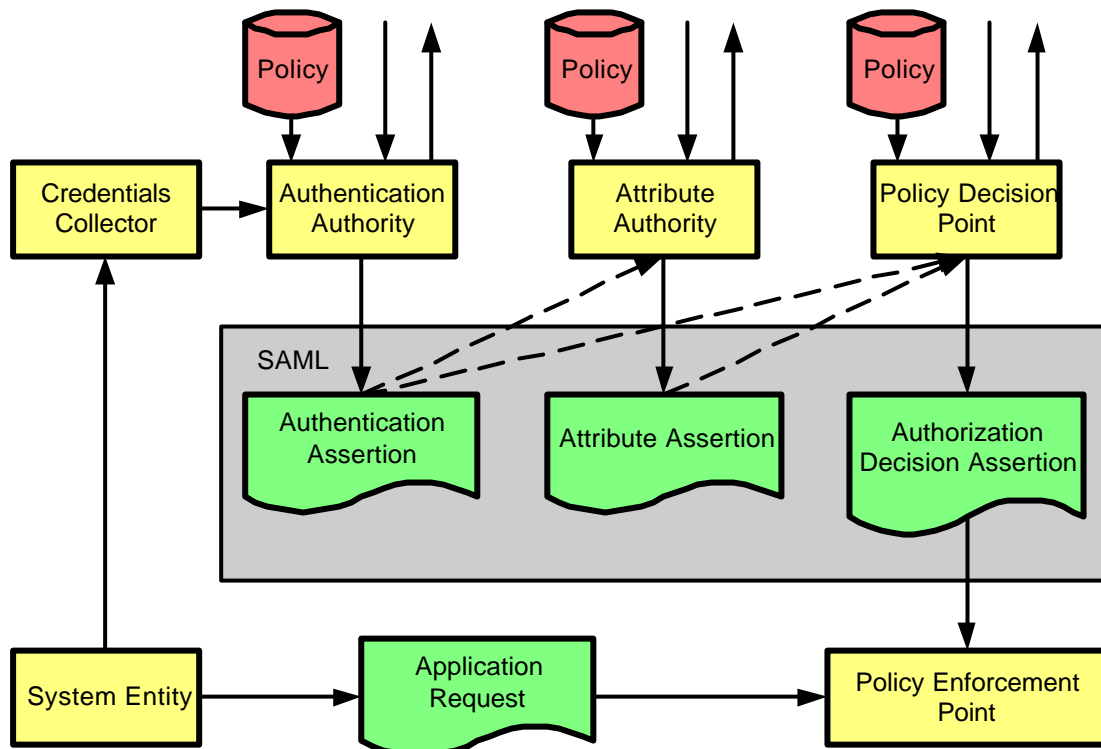


Figure 1 The SAML Domain Model

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

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

1.3.2. SAML and URI-Based Identifiers

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

urn:ietf:rfc:1510

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

http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/rwedc

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

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

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

330 **1.3.3. SAML and Extensibility**

331 The XML formats for SAML assertions and protocol messages have been designed to support
332 extension.

333 However, it is possible that the use of extensions will harm interoperability and the use of
334 extensions SHOULD be carefully considered. ~~[TBD] Need conceptual material here. Explain~~
335 ~~concepts/terms such as the domain model, SAML-defined namespaces, URIs for identifiers, what is~~
336 ~~out-of-band/scope, extension points, etc.~~

2. SAML Assertions

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

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

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

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

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

2.1. Schema Header and Namespace Declarations

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

```
<schema
  targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-
sstc-schema-assertion-24.xsd"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-
schema-assertion-24.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified">
  <import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="xmldsig-core-schema.xsd"/>
  <annotation>
    <documentation>draft-sstc-schema-assertion-24.xsd</documentation>
  </annotation>
  ...
</schema>
```

2.2. Simple Types

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

2.2.1. Simple Type IDType

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

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

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

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

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

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

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

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

2.2.2. Simple Type **DecisionType**

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

Permit

The specified action is permitted.

Deny

The specified action is denied.

Indeterminate

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

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

```
<simpleType name="DecisionType">
  <restriction base="string">
    <enumeration value="Permit"/>
    <enumeration value="Deny"/>
    <enumeration value="Indeterminate"/>
  </restriction>
</simpleType>
```

2.3. Assertions

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

2.3.1. Element **<AssertionSpecifier>**

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

<AssertionID>

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

<Assertion>

Specifies an assertion by value.

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

```
<element name="AssertionSpecifier" type="saml:AssertionSpecifierType"/>
<complexType name="AssertionSpecifierType">
  <choice>
    <element ref="saml:AssertionID"/>
    <element ref="saml:Assertion"/>
  </choice>
</complexType>
```

2.3.2. Element **<AssertionID>**

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

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

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

427 2.3.3. Element <Assertion>

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

431 MajorVersion [Required]

432 The major version of this assertion. The identifier for the version of SAML defined in this
433 specification is 1. Processing of this attribute is specified in Section [3.4.23-5.2](#).

434 MinorVersion [Required]

435 The minor version of this assertion. The identifier for the version of SAML defined in this
436 specification is 0. Processing of this attribute is specified in Section [3.4.23-5.2](#).

437 AssertionID [Required]

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

440 Issuer [Required]

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

444 IssueInstant [Required]

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

447 <Conditions> [Optional]

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

449 <Advice> [Optional]

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

452 One or more of the following statement elements:

453 <Statement>

454 A statement defined in an extension schema.

455 <SubjectStatement>

456 A subject statement defined in an extension schema.

457 <AuthenticationStatement>

458 An authentication statement.

459 <AuthorizationDecisionStatement>

460 An authorization decision statement.

461 <AttributeStatement>

462 An attribute statement.

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

```
465 <complexType name="AssertionType">  
466   <sequence>  
467     <element ref="saml:Conditions" minOccurs="0"/>  
468     <element ref="saml:Advice" minOccurs="0"/>  
469     <choice minOccurs="0" maxOccurs="unbounded">  
470       <element ref="saml:Statement"/>
```



```

471         <element ref="saml:SubjectStatement" />
472         <element ref="saml:AuthenticationStatement" />
473         <element ref="saml:AuthorizationDecisionStatement" />
474         <element ref="saml:AttributeStatement" />
475     </choice>
476     <element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded" />
477 </sequence>
478 <attribute name="MajorVersion" type="integer" use="required" />
479 <attribute name="MinorVersion" type="integer" use="required" />
480 <attribute name="AssertionID" type="saml:IDType" use="required" />
481 <attribute name="Issuer" type="string" use="required" />
482 <attribute name="IssueInstant" type="dateTime" use="required" />
483 </complexType>

```

2.3.3.1. Element <Conditions>

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

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

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

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

- NotBefore [Optional]
Specifies the earliest time instant at which the assertion is valid.
- NotOnOrAfter [Optional]
Specifies the time instant at which the assertion has expired.
- <Condition> [Zero or more]
Provides an extension point allowing extension schemas to define new conditions.
- <AudienceRestrictionCondition> [Any Number]
Specifies that the assertion is addressed to a particular audience.
- <TargetRestrictionCondition> [Any Number]
The <TargetRestriction> condition is used to limit the use of the assertion to a particular relying party.

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

```

511 <element name="Conditions" type="saml:ConditionsType" />
512 <complexType name="ConditionsType">
513     <choice minOccurs="0" maxOccurs="unbounded">
514         <element ref="saml:Condition" />
515         <element ref="saml:AudienceRestrictionCondition" />
516         <element ref="saml:TargetRestrictionCondition" />
517     </choice>
518     <attribute name="NotBefore" type="dateTime" use="optional" />
519     <attribute name="NotOnOrAfter" type="dateTime" use="optional" />

```

520 `</complexType>`

521 **2.3.3.1.1 Attributes *NotBefore* and *NotOnOrAfter***

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

523 The *NotBefore* attribute specifies the time instant at which the validity interval begins. The

524 *NotOnOrAfter* attribute specifies the time instant at which the validity interval has ended.

525 If the value for either *NotBefore* or *NotOnOrAfter* is omitted or is equal to the start of the epoch,
526 it is considered unspecified. If the *NotBefore* attribute is unspecified (and if any other conditions
527 that are supplied evaluate to *Valid*), the assertion is valid at any time before the time instant
528 specified by the *NotOnOrAfter* attribute. If the *NotOnOrAfter* attribute is unspecified (and if any
529 other conditions that are supplied evaluate to *Valid*), the assertion is valid from the time instant
530 specified by the *NotBefore* attribute with no expiry. If neither attribute is specified (and if any other
531 conditions that are supplied evaluate to *Valid*), the assertion is valid at any time.

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

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

536 **2.3.3.1.2 Element *<Condition>***

537 The *<Condition>* element serves as an extension point for new conditions. Its

538 **ConditionAbstractType** complex type is abstract; extension elements MUST use the *xsi:type*
539 attribute to indicate the derived type.

540 The following schema fragment defines the *<Condition>* element and its

541 **ConditionAbstractType** complex type:

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

544 **2.3.3.1.3 Elements *<AudienceRestrictionCondition>* and *<Audience>***

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

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

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

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

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

```
561 <element name="AudienceRestrictionCondition"  
562         type="saml:AudienceRestrictionConditionType"/>  
563 <complexType name="AudienceRestrictionConditionType">  
564     <complexContent>
```

```

565         <extension base="saml:ConditionAbstractType">
566             <sequence>
567                 <element ref="saml:Audience"
568                     minOccurs="1" maxOccurs="unbounded" />
569             </sequence>
570         </extension>
571     </complexContent>
572 </complexType>
573 <element name="Audience" type="anyURI" />

```

574 2.3.3.1.4 Condition Type TargetRestrictionType

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

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

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

```

581 <element name="TargetRestrictionCondition"
582     type="saml:TargetRestrictionConditionType" />
583 <complexType name="TargetRestrictionConditionType">
584     <complexContent>
585         <extension base="saml:ConditionAbstractType">
586             <sequence>
587                 <element ref="saml:Target"
588                     minOccurs="1" maxOccurs="unbounded" />
589             </sequence>
590         </extension>
591     </complexContent>
592 </complexType>
593 <element name="Target" type="anyURI" />

```

594 2.3.3.2. Elements <Advice> and <AdviceElement>

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

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

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

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

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

```

608 <element name="Advice" type="saml:AdviceType" />
609 <complexType name="AdviceType">
610     <sequence>
611         <choice minOccurs="0" maxOccurs="unbounded">
612             <element ref="saml:AssertionSpecifier"/>
613             <element ref="saml:AdviceElement"/>
614             <any namespace="##other" processContents="lax" />

```

```

615         </choice>
616     </sequence>
617 </complexType>
618 <element name="AdviceElement" type="saml:AdviceAbstractType" />
619 <complexType name="AdviceAbstractType"/>

```

2.4. Statements

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

2.4.1. Element <Statement>

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

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

```

628 <element name="Statement" type="saml:StatementAbstractType" />
629 <complexType name="StatementAbstractType" abstract="true" />

```

2.4.2. Element <SubjectStatement>

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

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

```

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

```

2.4.2.1. Element <Subject>

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

<NameIdentifier>

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

<SubjectConfirmation>

Information that allows the subject to be authenticated.

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

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

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

672 2.4.2.2. Element <NameIdentifier>

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

675 SecurityDomain

676 The security domain governing the name of the subject.

677 Name

678 The name of the subject.

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

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

```
684 <element name="NameIdentifier" type="saml:NameIdentifierType"/>
685 <complexType name="NameIdentifierType">
686   <attribute name="SecurityDomain" type="string"/>
687   <attribute name="Name" type="string"/>
688 </complexType>
```

689 2.4.2.3. Elements <SubjectConfirmation>, <ConfirmationMethod>, and 690 <SubjectConfirmationData>

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

693 <ConfirmationMethod> [One or more]

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

696 <SubjectConfirmationData> [Zero or more]

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

698 <ds:KeyInfo> [Optional]

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

701 The following schema fragment defines the <SubjectConfirmation> element and its
702 **SubjectConfirmationType** complex type, along with the <SubjectConfirmationData>
703 element and the <ConfirmationMethod> element:

```
704 <element name="SubjectConfirmation" type="saml:SubjectConfirmationType"/>
705 <complexType name="SubjectConfirmationType">
706   <sequence>
707     <element ref="saml:ConfirmationMethod" maxOccurs="unbounded"/>
708     <element ref="saml:SubjectConfirmationData" minOccurs="0"/>
```

```

709     <element ref="ds:KeyInfo" minOccurs="0"/>
710   </sequence>
711 </complexType>
712 <element name="SubjectConfirmationData" type="string" minOccurs="0"/>
713 <element name="ConfirmationMethod" type="anyURI"/>

```

2.4.3. Element <AuthenticationStatement>

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

AuthenticationMethod [Required]

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

AuthenticationInstant [Required]

Specifies the time at which the authentication took place.

<AuthenticationLocality> [Optional]

Specifies ~~the DNS domain name and IP address for the system entity that was authenticated. the DNS domain name and IP address for the system entity that performed the authentication.~~

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

```

730 <element name="AuthenticationStatement"
731       type="saml:AuthenticationStatementType"/>
732 <complexType name="AuthenticationStatementType">
733   <complexContent>
734     <extension base="saml:SubjectStatementAbstractType">
735       <sequence>
736         <element ref="saml:AuthenticationLocality" minOccurs="0"/>
737         <element ref="saml:AuthorityBinding"
738               minOccurs="0" maxOccurs="unbounded"/>
739       </sequence>
740       <attribute name="AuthenticationMethod" type="anyURI"/>
741       <attribute name="AuthenticationInstant" type="dateTime"/>
742     </extension>
743   </complexContent>
744 </complexType>

```

2.4.3.1. Element <AuthenticationLocality>

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

IPAddress [Optional]

The IP address of the system entity that was authenticated.

DNSAddress [Required]

The DNS address of the system entity that was authenticated.

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

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

```

756 <element name="AuthenticationLocality"
757       type="saml:AuthenticationLocalityType"/>

```

```

<complexType name="AuthenticationLocalityType">
  <attribute name="IPAddress" type="string" use="optional"/>
  <attribute name="DNSAddress" type="string" use="optional"/>
</complexType>

```

2.4.3.2. Element <AuthorityBinding>

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

AuthorityKind [Optional]

The type of authority that performed the authentication.

Binding [Optional]

The address of the authority.

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

```

<element name="AuthorityBinding" type="saml:AuthorityBindingType"/>
<complexType name="AuthorityBindingType">
  <attribute name="AuthorityKind" type="saml:AuthorityKindType"/>
  <attribute name="Binding" type="anyURI"/>
</complexType>
<simpleType name="AuthorityKindType">
  <restriction base="string">
    <enumeration value="authentication"/>
    <enumeration value="attribute"/>
    <enumeration value="authorization"/>
  </restriction>
</simpleType>

```

2.4.4. Element <AuthorizationDecisionStatement>

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

AuthorizationDecisionStatementType, which extends **SubjectStatementAbstractType** with the addition of the following elements (in order) and attributes:

Resource [Optional]

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

Decision [Optional]

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

<Actions> [Required]

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

<Evidence> [Zero or more]

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

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

```

<element name="AuthorizationDecisionStatement"
  type="saml:AuthorizationDecisionStatementType"/>
<complexType name="AuthorizationDecisionStatementType">
  <complexContent>
    <extension base="saml:SubjectStatementAbstractType">
      <sequence>
        <element ref="saml:Actions"/>

```



```

      <element ref="saml:Evidence" minOccurs="0"
        maxOccurs="unbounded"/>
    </sequence>
    <attribute name="Resource" type="anyURI" use="optional"/>
    <attribute name="Decision" type="saml:DecisionType"
      use="optional"/>
  </extension>
</complexContent>
</complexType>

```

2.4.4.1. Elements <Actions> and <Action>

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

Namespace [Optional]

A URI representing the namespace in which the names of specified actions are to be interpreted. If this element is absent, the namespace <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/rwcdc-negation> specified in section 7.2.2 is in effect by default.

<Action> [One or more]

An action sought to be performed on the specified resource.

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

```

<element name="Actions" type="saml:ActionsType"/>
<complexType name="ActionsType">
  <sequence>
    <element ref="saml:Action" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Namespace" type="anyURI" use="optional"/>
</complexType>
<element name="Action" type="string"/>

```

2.4.4.2. Element <Evidence>

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

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

The following schema fragment defines the <Evidence> element:

```

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

```

2.4.5. Element <AttributeStatement>

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

<Attribute> [One or More]

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

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

```

<element name="AttributeStatement" type="saml:AttributeStatementType"/>

```



```

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

```

863 2.4.5.1. Elements <AttributeDesignator> and <Attribute>

864 The <AttributeDesignator> element identifies an attribute name within an attribute
865 namespace. It has the **AttributeDesignatorType** complex type. It is used in an attribute assertion
866 query to request that attribute values within a specific namespace be returned (see [3.3.43-4.4](#) for
867 more information). The <AttributeDesignator> element contains the following XML attributes:

868 AttributeNamespace [Required]

869 The namespace in which the AttributeName elements are interpreted.

870 AttributeName [Required]

871 The name of the attribute.

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

```

874     <element name="AttributeDesignator" type="saml:AttributeDesignatorType" />
875     <complexType name="AttributeDesignatorType">
876       <attribute name="AttributeName" type="string" />
877       <attribute name="AttributeNamespace" type="anyURI" />
878     </complexType>

```

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

882 <AttributeValue> [Required]

883 The value of the attribute.

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

```

886     <element name="Attribute" type="saml:AttributeType" />
887     <complexType name="AttributeType">
888       <complexContent>
889         <extension base="saml:AttributeDesignatorType">
890           <sequence>
891             <element ref="saml:AttributeValue" />
892           </sequence>
893         </extension>
894       </complexContent>
895     </complexType>

```

896 2.4.5.1.1 Element <AttributeValue>

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

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

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

```
906 <element name="AttributeValue" type="saml:AttributeValueType"/>
907 <complexType name="AttributeValueType">
908   <sequence>
909     <any namespace="##any" processContents="lax"
910       minOccurs="0" maxOccurs="unbounded" />
911   </sequence>
912 </complexType>
```

3. SAML Protocol

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

SAML-aware clients MAY in addition use the SAML request-response protocol defined by the <Request> and <Response> elements. The client sends a <Request> element to a SAML service, and the service generates a <Response> element, as shown in Figure 2Figure 4.



Figure 24: SAML Request-Response Protocol

3.1. Schema Header and Namespace Declarations

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

```
<schema
  targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-
sstc-schema-protocol-24.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:samlp="http://www.oasis-open.org/committees/security/docs/draft-sstc-
schema-protocol-24.xsd"
  xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-
schema-assertion-24.xsd"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  elementFormDefault="unqualified">
  <import namespace="http://www.oasis-open.org/committees/security/docs/draft-
sstc-schema-assertion-24.xsd"
    schemaLocation="draft-sstc-schema-assertion-24.xsd" />
  <import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="xmldsig-core-schema.xsd" />
  <annotation>
    <documentation>draft-sstc-schema-protocol-24.xsd</documentation>
  </annotation>
  ...
</schema>
```

3.2. Simple Types

~~The following sections define the SAML protocol-related simple types.~~

3.2.1. Simple Type StatusCodeType

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

~~Success~~

~~The request succeeded.~~

~~Failure~~

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

~~Error~~

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

~~Unknown~~
~~The request failed for unknown reasons.~~
~~The following schema fragment defines the **StatusCodeType** simple type:~~

```
<simpleType name="StatusCodeType">  
  <restriction base="string">  
    <enumeration value="Success"/>  
    <enumeration value="Failure"/>  
    <enumeration value="Error"/>  
    <enumeration value="Unknown"/>  
  </restriction>  
</simpleType>
```

3.2. Requests

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

3.2.1. Complex Type RequestAbstractType

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

RequestID [Required]

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

MajorVersion [Required]

The major version of this request. The identifier for the version of SAML defined in this specification is 1. Processing of this attribute is specified in Section [3.4.23.5.2](#).

MinorVersion [Required]

The minor version of this request. The identifier for the version of SAML defined in this specification is 0. Processing of this attribute is specified in Section [3.4.23.5.2](#).

<RespondWith> [Any Number]

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

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

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

3.2.1.1. Element <RespondWith>

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

SingleStatement

An assertion carrying exactly one statement element.

1002 `MultipleStatement`
 1003 An assertion carrying at least one statement element.

1004 `AuthenticationStatement`
 1005 An assertion carrying an Authentication statement.

1006 `AuthorizationDecisionStatement`
 1007 An assertion carrying an Authorization Decision statement.

1008 `AttributeStatement`
 1009 An assertion carrying an Attribute statement.

1010 *Schema URI*
 1011 An assertion containing additional elements from the specified schema.

1012 The following schema fragment defines the `<RespondWith>` element:

```
1013 <element name="RespondWith" type="anyURI" />
```

1014 3.2.2. Element `<Request>`

1015 The `<Request>` element specifies a SAML request. It provides either a query or a request for a
 1016 specific assertion identified by `<AssertionID>` or `<AssertionArtifact>`. It has the complex
 1017 type **RequestType**, which extends **RequestAbstractType** by adding a choice of one of the
 1018 following elements:

1019 `<Query>`
 1020 An extension point that allows extension schemas to define new types of query.

1021 `<SubjectQuery>`
 1022 An extension point that allows extension schemas to define new types of query that specify
 1023 a single SAML subject.

1024 `<AuthenticationQuery>`
 1025 Makes a query for authentication information.

1026 `<AttributeQuery>`
 1027 Makes a query for attribute information.

1028 `<AuthorizationDecisionQuery>`
 1029 Makes a query for an authorization decision.

1030 `<AssertionID>` [One or more]
 1031 Requests an assertion by reference to its assertion identifier.

1032 `<AssertionArtifact>` [One or more]
 1033 Requests an assertion by supplying an assertion artifact that represents it.

1034 The following schema fragment defines the `<Request>` element and its **RequestType** complex
 1035 type:

```
1036 <element name="Request" type="samlp:RequestType"/>
1037 <complexType name="RequestType">
1038   <complexContent>
1039     <extension base="samlp:RequestAbstractType">
1040       <choice>
1041         <element ref="samlp:Query"/>
1042         <element ref="samlp:SubjectQuery"/>
1043         <element ref="samlp:AuthenticationQuery"/>
1044         <element ref="samlp:AttributeQuery"/>
1045         <element ref="samlp:AuthorizationDecisionQuery"/>
1046         <element ref="saml:AssertionID" maxOccurs="unbounded"/>
1047         <element ref="samlp:AssertionArtifact" maxOccurs="unbounded"/>
1048       </choice>
1049     </extension>
```

```

1050     </complexContent>
1051   </complexType>
1052   <element name="AssertionArtifact" type="string"/>

```

3.3. Queries

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

3.3.1. Element <Query>

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

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

```

1062   <element name="Query" type="samlp:QueryAbstractType"/>
1063   <complexType name="QueryAbstractType" abstract="true"/>

```

3.3.2. Element <SubjectQuery>

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

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

```

1071   <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType"/>
1072   <complexType name="SubjectQueryAbstractType" abstract="true">
1073     <complexContent>
1074       <extension base="samlp:QueryAbstractType">
1075         <sequence>
1076           <element ref="saml:Subject"/>
1077         </sequence>
1078       </extension>
1079     </complexContent>
1080   </complexType>

```

3.3.3. Element <AuthenticationQuery>

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

<ConfirmationMethod> [Optional]

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

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

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

```
<element name="AuthenticationQuery" type="samlp:AuthenticationQueryType"/>
<complexType name="AuthenticationQueryType">
  <complexContent>
    <extension base="samlp:SubjectQueryAbstractType">
      <sequence>
        <element ref="saml:ConfirmationMethod" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

3.3.4. Element <AttributeQuery>

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

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

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

```
<element name="AttributeQuery" type="samlp:AttributeQueryType"/>
<complexType name="AttributeQueryType">
  <complexContent>
    <extension base="samlp:SubjectQueryAbstractType">
      <sequence>
        <element ref="saml:AttributeDesignator"
          minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="CompletenessSpecifier"
        type="samlp:CompletenessSpecifierType" use="required"/>
    </extension>
  </complexContent>
</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?” The response will be in the form of an assertion containing an authorization decision statement. This element is of type **AuthorizationDecisionQueryType**, which extends **SubjectQueryAbstractType** with the addition of the following elements and attribute:

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

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

<Evidence> [Zero or more]
An assertion that the responder MAY rely on in making its response.

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

```

1144     <element name="AuthorizationDecisionQuery "
1145 type="samlp:AuthorizationDecisionQueryType"/>
1146   <complexType name="AuthorizationDecisionQueryType">
1147     <complexContent>
1148       <extension base="samlp:SubjectQueryAbstractType">
1149         <sequence>
1150           <element ref="saml:Actions"/>
1151           <element ref="saml:Evidence"
1152             minOccurs="0" maxOccurs="unbounded" />
1153         </sequence>
1154         <attribute name="Resource" type="anyURI" />
1155       </extension>
1156     </complexContent>
1157   </complexType>

```

3.4. Responses

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

3.4.1. Complex Type ResponseAbstractType

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

ResponseID [Required]

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

InResponseTo [Required]

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

MajorVersion [Required]

The major version of this response. The identifier for the version of SAML defined in this specification is 1. Processing of this attribute is specified in Section [3.4.23.5.2](#).

MinorVersion [Required]

The minor version of this response. The identifier for the version of SAML defined in this specification is 0. Processing of this attribute is specified in Section [3.4.23.5.2](#).

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

```

1176   <complexType name="ResponseAbstractType" abstract="true" >
1177     <sequence>
1178       <element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded" />
1179     </sequence>
1180     <attribute name="ResponseID" type="saml:IDType" use="required" />
1181     <attribute name="InResponseTo" type="saml:IDType" use="required" />
1182     <attribute name="MajorVersion" type="integer" use="required" />
1183     <attribute name="MinorVersion" type="integer" use="required" />
1184   </complexType>

```

3.4.2. Element <Response>

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

<Status> ~~Code~~ [Required] (see Section [3.4.33.2.4](#))

A code representing the status of the corresponding request.

<Assertion> (see Section 2.3.3)
Specifies an assertion by value.

~~<SingleAssertion>
Specifies an assertion containing a single statement by value.~~

~~<MultipleAssertion>
Specifies an assertion containing multiple statements by value.~~

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

```
<element name="Response" type="samlp:ResponseType" />
<complexType name="ResponseType">
  <complexContent>
    <extension base="samlp:ResponseAbstractType">
      <sequence>
        <element ref="samlp:Status"/>
        <element ref="saml:Assertion"
          minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </extension>
    <extension base="samlp:ResponseAbstractType">
      <sequence>
        <element ref="samlp:StatusReason"
          minOccurs="0" maxOccurs="unbounded"/>
        <element ref="saml:Assertion"
          minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="StatusCode"
        type="samlp:StatusCodeType" use="required"/>
    </extension>
  </complexContent>
</complexType>
```

3.4.3. Element <Status>

The <Status> element:

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

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

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

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

```
<element name="Status" type="samlp:StatusType" />
<complexType name="StatusType">
  <sequence>
    <element ref="samlp:StatusCode"/>
    <element ref="samlp:StatusMessage"
      minOccurs="0" maxOccurs="unbounded"/>
    <element ref="samlp:StatusDetail" minOccurs="0"/>
  </sequence>
</complexType>
```

3.4.3.1. Element <StatusCode>

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

1242 Value [Required]
 1243 The status code value as defined below.

1244 SubStatusCode [Optional]
 1245 An optional status code value that provides more specific information on an error condition.

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

1247 Success
 1248 The request succeeded.

1249 VersionMismatch
 1250 The receiver could not process the request because the version was incorrect.

1251 Receiver
 1252 The request could not be performed due to an error at the receiving end.

1253 Sender
 1254 The request could not be performed due to an error in the sender or in the request

1255 The following **SubStatusCode** values are defined, additional codes MAY be defined in future
 1256 versions of the SAML specification:

1257 RequestVersionTooHigh
 1258 The protocol version specified in the request is a major upgrade from the highest protocol
 1259 version supported by the responder.

1260 RequestVersionTooLow
 1261 The responder cannot respond to the particular request using the SAML version specified
 1262 in the request because it is too low.

1263 RequestVersionDeprecated
 1264 The responder does not respond to any requests with the protocol version specified in the
 1265 request.

1266 TooManyResponses
 1267 The response would contain more elements than the responder will return.

1268 The following schema fragment defines the <StatusCode> element and its **StatusCodeType**
 1269 complex type and the **StatusCodeEnumType** simple type:

```

1270 <element name="StatusCode" type="samlp:StatusCodeType"/>
1271 <complexType name="StatusCodeType">
1272 <attribute name="Value" type="samlp:StatusCodeEnumType"/>
1273 <attribute name="SubStatusCode" type="QName" use="optional"/>
1274 </complexType>
1275 <simpleType name="StatusCodeEnumType">
1276 <restriction base="QName">
1277 <enumeration value="samlp:Success"/>
1278 <enumeration value="samlp:VersionMismatch"/>
1279 <enumeration value="samlp:Receiver"/>
1280 <enumeration value="samlp:Sender"/>
1281 </restriction>
1282 </simpleType>

```

1283 **3.4.3.2. Element <StatusMessage>**

1284 The <StatusMessage> element specifies a message that MAY be returned to an operator:

1285 The following schema fragment defines the <StatusMessage> element and its
 1286 **StatusMessageType** complex type:

```

1287 <element name="StatusMessage" type="string"/>

```

3.4.3.3. Element <StatusDetail>

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

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

```
<element name="StatusDetail" type="samlp:StatusDetailType"/>
<complexType name="StatusDetailType">
  <sequence>
    <any namespace="##any"
      processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

3.5.2.1. Element <StatusReason>

The <StatusReason> element provides additional information that indicates the reason for the return of an Error or Failure status code. The following values are defined. Implementations MAY define additional codes:

~~RequestVersionTooHigh~~

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

~~RequestVersionTooLow~~

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

~~RequestVersionDeprecated~~

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

~~TooManyResponses~~

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

The following schema fragment defines the <StatusReason> element:

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

3.4.4. Simple Type StatusCodeType

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

Success

The request succeeded.

Failure

The request could not be performed by the service.

Error

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

Unknown

The request failed for unknown reasons.

The following schema fragment defines the StatusCodeType simple type:

```
<simpleType name="StatusCodeType">
  <restriction base="string">
    <enumeration value="Success"/>
    <enumeration value="Failure"/>
    <enumeration value="Error"/>
  </restriction>
</simpleType>
```

1334

|

<enumeration value="Unknown"/>

</restriction>

</simpleType>

4. SAML Versioning

SAML version information appears in the following elements:

?? <Assertion>

?? <Request>

?? <Response>

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

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

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

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

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

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

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

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

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

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

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

4.1. Assertion Version

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

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

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

4.2. Request Version

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

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

4.3. Response Version

1375

1376 A SAML application MUST NOT issue responses that specify a higher SAML version number than
1377 the corresponding request.

1378 A SAML application MUST NOT issue a response that has a major version number that is lower
1379 than the major version number of the corresponding request except to report the error

1380 `RequestVersionTooHigh`.

1381 Incompatible protocol versions MAY cause the following errors to be reported:

1382 `RequestVersionTooHigh`

1383 The protocol version specified in the request is a major upgrade from the highest protocol
1384 version supported by the responder.

1385 `RequestVersionTooLow`

1386 The responder cannot respond to the particular request using the SAML version specified
1387 in the request because it is too low.

1388 `RequestVersionDeprecated`

1389 The responder does not respond to any requests with the protocol version specified in the
1390 request.

5. SAML & XML-Signature Syntax and Processing

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

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

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

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

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

Note :

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

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

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

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

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

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

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

- ~~(2)~~ SAML message obtained arriving at a destination from an entity other than the originating site MUST be signed by the origin site.

5.1. Signing Assertions

All SAML assertions MAY be signed using the XML Signature. This is reflected in the assertion schema – Section 2.3.3.

5.2. Request/Response Signing

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

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

5.3.1. Rationale

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

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

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

- ~~(2)~~ A SAML request or response may be viewed as inheriting a signature from a super signature, if the super signature applies to all the elements within the response.

5.3.2. Rules for SAML Signature Inheritance

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

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

5.4. XML Signature Profile

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

5.4.1. Signing formats

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

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

5.4.2. CanonicalizationMethod

~~XML Signature~~~~[Sig]~~ 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.

5.4.3. Transforms

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

5.4.4. KeyInfo

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

5.4.5. Binding between statements in a multi-statement assertion

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

5.4.6. Security considerations

5.4.6.1. Replay Attack

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

6. SAML Extensions

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

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

6.1. Assertion Schema Extension

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

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

?? <Assertion>

?? <Condition>

?? <Statement>

?? <SubjectStatement>

?? <AdviceElement>

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

~~?? <SingleAssertion>~~

~~?? <MultipleAssertion>~~

?? <AuthenticationStatement>

?? <AuthorizationDecisionStatement>

?? <AttributeStatement>

?? <AudienceRestrictionCondition>

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

?? <AttributeValue>

?? <Advice>

6.2. Protocol Schema Extension

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

?? <Query>

?? <SubjectQuery>

1548 In addition, the following elements that are directly usable as part of SAML MAY be extended:

1549 ?? <Request>

1550 ?? <AuthenticationQuery>

1551 ?? <AuthorizationDecisionQuery>

1552 ?? <AttributeQuery>

1553 ?? <Response>

1554 6.3. Use of Type Derivation and Substitution Groups

1555 W3C XML Schema [Schema1] provides two principal mechanisms for specifying an element of an
1556 extended type: type derivation and substitution groups.

1557 For example, a <Statement> element can be assigned the type **NewStatementType** by means of
1558 the `xsi:type` attribute. For such an element to be schema-valid, **NewStatementType** needs to be
1559 derived from **StatementType**. The following example of a SAML assertion assumes that the
1560 extension schema (represented by the `new:` prefix) has defined this new type:

```
1561 <saml:Assertion ...>  
1562   <saml:Statement xsi:type="new:NewStatementType">  
1563     ...  
1564   </saml:Statement>  
1565 </saml:Assertion>
```

1566 Alternatively, the extension schema can define a <NewStatement> element that is a member of a
1567 substitution group that has <Statement> as a head element. For the substituted element to be
1568 schema-valid, it needs to have a type that matches or is derived from the head element's type. The
1569 following is an example of an extension schema fragment that defines this new element:

```
1570 <xsd:element "NewStatement" type="new:NewStatementType"  
1571   substitutionGroup="saml:Statement"/>
```

1572 The substitution group declaration allows the <NewStatement> element to be used anywhere the
1573 SAML <Statement> element can be used. The following is an example of a SAML assertion that
1574 uses the extension element:

```
1575 <saml:Assertion ...>  
1576   <new:NewStatement>  
1577     ...  
1578   </new:NewStatement>  
1579 </saml:Assertion>
```

1580 The choice of extension method has no effect on the semantics of the XML document but does
1581 have implications for interoperability.

1582 The advantages of type derivation are as follows:

1583 ?? A document can be more fully interpreted by a parser that does not have access to the
1584 extension schema because a "native" SAML element is available.

1585 ?? At the time of writing, some W3C XML Schema validators do not support substitution
1586 groups, whereas the `xsi:type` attribute is widely supported.

1587 The advantage of substitution groups is that a document can be explained without the need to
1588 explain the functioning of the `xsi:type` attribute.

7. SAML-Defined Identifiers

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

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

<http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/>

7.1. Confirmation Method Identifiers

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

7.1.1. SAML Artifact:

URI: <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/artifact>

<SubjectConfirmationData>: *Base64 (Artifact)*

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

7.1.2. SAML Artifact (SHA-1):

URI: <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/artifact-sha1>

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

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

7.1.3. Holder of Key:

URI: <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/Holder-Of-Key>

<ds:KeyInfo>: Any cryptographic key

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

7.1.4. Sender Vouches:

URI: <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/sender-vouches>

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

7.1.5. Password (Pass-Through):

URI: <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/password>

<SubjectConfirmationData>: *Base64 (Password)*

1621 The subject of the assertion is the party that can present the password value specified in
1622 <SubjectConfirmationData>.

1623 The username of the subject is specified by means of the <NameIdentifier> element.

1624 **7.1.6. Password (One-Way-Function SHA-1):**

1625 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/password-sha1>

1626 <SubjectConfirmationData>: *Base64 (SHA1 (Password))*

1627 The subject of the assertion is the party that can present the password such that the SHA1 digest of
1628 the specified password matches the value specified in <SubjectConfirmationData>.

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

1630 **7.1.7. Kerberos** ~~[Kerberos]~~

1631 **URI:** <urn:ietf:rfc:1510>

1632 <SubjectConfirmationData>: A Kerberos Ticket

1633 ~~The subject is authenticated by means of the Kerberos protocol [RFC 1510], an instantiation of the~~
1634 ~~Needham-Schroeder symmetric key authentication mechanism [Needham78].~~

1635 **7.1.8. SSL/TLS Certificate Based Client Authentication:**

1636 **URI:** <urn:ietf:rfc:2246>

1637 <ds:KeyInfo>: Any cryptographic key

1638 **7.1.9. Object Authenticator (SHA-1):**

1639 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/object-sha1>

1640 <SubjectConfirmationData>: *Base64 (SHA1 (Object))*

1641 This authenticator element is the result of computing a digest, using the SHA-1 hash algorithm. It is
1642 used when the subject can be represented as a binary string, for example when it is an XML
1643 document or the disk image of executable code. Any preprocessing of the subject prior to
1644 computation of the digest is out of scope. The name of the subject should be conveyed in an
1645 accompanying NameIdentifier element.

1646 **7.1.10. PKCS#7**

1647 **URI:** <urn:ietf:rfc:2315>

1648 <SubjectConfirmationData>: *Base64 (PKCS#7 (Object))*

1649 This authenticator element is signed data in PKCS#7 format [PKCS#7]. The posited identity of the
1650 signer must be conveyed in an accompanying NameIdentifier element. This subject type may be
1651 included in the subject field of an authentication query, in which case the corresponding response
1652 indicates whether the posited signer is, indeed, the signer. It may be included in an attribute query,
1653 in which case, the requested attribute values for the subject authenticated by the signed data are
1654 returned. It may be included in an authorization query, in which case, the access request
1655 represented by the signed data shall be identified by the accompanying object element, and the

1656 corresponding authorization decision assertion indicates whether the signer is authorized for the
1657 access request represented by the object element.

1658 **7.1.11. Cryptographic Message Syntax**

1659 **URI:** urn:ietf:rfc:2630

1660 `<SubjectConfirmationData>`: *Base64* (CMS (*Object*))

1661 This authenticator element is signed data in CMS format [CMS]. See also 7.1.10

1662 **7.1.12. XML Digital Signature**

1663 **URI:** urn:ietf:rfc:2630

1664 `<SubjectConfirmationData>`: *Base64* (XML-SIG (*Object*))

1665 `<ds:KeyInfo>`: A cryptographic signing key

1666 This authenticator element is signed data in XML Signature format. See also 7.1.10

1667 **7.2. Action Namespace Identifiers**

1668 The following identifiers MAY be used in the `ActionNamespace` attribute (see Section 2.4.4.1) to
1669 refer to common sets of actions to perform on resources.

1670 **7.2.1. Read/Write/Execute/Delete/Control:**

1671 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/rwedc>

1672 Defined actions:

1673 Read Write Execute Delete Control

1674 These actions are interpreted in the normal manner, i.e.

1675 Read

1676 The subject may read the resource

1677 Write

1678 The subject may modify the resource

1679 Execute

1680 The subject may execute the resource

1681 Delete

1682 The subject may delete the resource

1683 Control

1684 The subject may specify the access control policy for the resource

1685 **7.2.2. Read/Write/Execute/Delete/Control with Negation:**

1686 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/rwedc-negation>

1687 Defined actions:

1688 Read Write Execute Delete Control ~Read ~Write ~Execute ~Delete ~Control

1689 The actions specified in section 7.2.1 are interpreted in the same manner described there. Actions
1690 prefixed with a tilde ~ are negated permissions and are used to affirmatively specify that the stated
1691 permission is denied. Thus a subject described as being authorized to perform the action ~Read is
1692 affirmatively denied read permission.

1693 An application MUST NOT authorize both an action and its negated form.

1694 **7.2.3. Get/Head/Put/Post:**

1695 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/ghpp>

1696 Defined actions:

1697 GET HEAD PUT POST

1698 These actions bind to the corresponding HTTP operations. For example a subject authorized to
1699 perform the GET action on a resource is authorized to retrieve it.

1700 The GET and HEAD actions loosely correspond to the conventional read permission and the PUT
1701 and POST actions to the write permission. The correspondence is not exact however since a HTTP
1702 GET operation may cause data to be modified and a POST operation may cause modification to a
1703 resource other than the one specified in the request. For this reason a separate Action URI
1704 specifier is provided.

1705 **7.2.4. UNIX File Permissions:**

1706 **URI:** <http://www.oasis-open.org/committees/security/docs/draft-sstc-core-24/unix>

1707 The defined actions are the set of UNIX file access permissions expressed in the numeric (octal)
1708 notation.

1709 The action string is a four digit numeric code:

1710 *extended user group world*

1711 Where the *extended* access permission has the value

1712 +2 if sgid is set

1713 +4 if suid is set

1714 The *user group* and *world* access permissions have the value

1715 +1 if execute permission is granted

1716 +2 if write permission is granted

1717 +4 if read permission is granted

1718 For example 0754 denotes the UNIX file access permission: user read, write and execute, group
1719 read and execute and world read.

8. SAML Schema Listings

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

8.1. Assertion Schema

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-Baker
(VeriSign Inc.) -->
<schema
  targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-
sstc-schema-assertion-24.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema" xmlns:saml="http://www.oasis-
open.org/committees/security/docs/draft-sstc-schema-assertion-24.xsd"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  elementFormDefault="unqualified">
  <import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="xmldsig-core-schema.xsd"/>
  <annotation>
    <documentation>draft-sstc-schema-assertion-24.xsd</documentation>
  </annotation>
  <simpleType name="IDType">
    <restriction base="string"/>
  </simpleType>
  <simpleType name="DecisionType">
    <restriction base="string">
      <enumeration value="Permit"/>
      <enumeration value="Deny"/>
      <enumeration value="Indeterminate"/>
    </restriction>
  </simpleType>
  <element name="AssertionSpecifier" type="saml:AssertionSpecifierType"/>
  <complexType name="AssertionSpecifierType">
    <choice>
      <element ref="saml:AssertionID"/>
      <element ref="saml:Assertion"/>
    </choice>
  </complexType>
  <element name="AssertionID" type="saml:IDType"/>
  <element name="Assertion" type="saml:AssertionType"/>
  <complexType name="AssertionType">
    <sequence>
      <element ref="saml:Conditions" minOccurs="0"/>
      <element ref="saml:Advice" minOccurs="0"/>
      <choice minOccurs="0" maxOccurs="unbounded">
        <element ref="saml:Statement"/>
        <element ref="saml:SubjectStatement"/>
        <element ref="saml:AuthenticationStatement"/>
        <element ref="saml:AuthorizationDecisionStatement"/>
        <element ref="saml:AttributeStatement"/>
      </choice>
      <element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="MajorVersion" type="integer" use="required"/>
    <attribute name="MinorVersion" type="integer" use="required"/>
    <attribute name="AssertionID" type="saml:IDType" use="required"/>
    <attribute name="Issuer" type="string" use="required"/>
    <attribute name="IssueInstant" type="dateTime" use="required"/>
  </complexType>
```



```

1777 <element name="Conditions" type="saml:ConditionsType"/>
1778 <complexType name="ConditionsType">
1779   <choice minOccurs="0" maxOccurs="unbounded">
1780     <element ref="saml:Condition"/>
1781     <element ref="saml:AudienceRestrictionCondition"/>
1782   </choice>
1783   <attribute name="NotBefore" type="dateTime" use="optional"/>
1784   <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>
1785 </complexType>
1786 <element name="Condition" type="saml:ConditionAbstractType"/>
1787 <complexType name="ConditionAbstractType" abstract="true"/>
1788 <element name="AudienceRestrictionCondition"
1789   type="saml:AudienceRestrictionConditionType"/>
1790 <complexType name="AudienceRestrictionConditionType">
1791   <complexContent>
1792     <extension base="saml:ConditionAbstractType">
1793       <sequence>
1794         <element ref="saml:Audience" maxOccurs="unbounded"/>
1795       </sequence>
1796     </extension>
1797   </complexContent>
1798 </complexType>
1799 <element name="Audience" type="anyURI"/>
1800 <element name="TargetRestrictionCondition"
1801   type="saml:TargetRestrictionConditionType"/>
1802 <complexType name="TargetRestrictionConditionType">
1803   <complexContent>
1804     <extension base="saml:ConditionAbstractType">
1805       <sequence>
1806         <element ref="saml:Target"
1807           minOccurs="1" maxOccurs="unbounded"/>
1808       </sequence>
1809     </extension>
1810   </complexContent>
1811 </complexType>
1812 <element name="Target" type="anyURI"/>
1813 <element name="Advice" type="saml:AdviceType"/>
1814 <complexType name="AdviceType">
1815   <sequence>
1816     <choice minOccurs="0" maxOccurs="unbounded">
1817       <element ref="saml:AssertionSpecifier"/>
1818       <element ref="saml:AdviceElement"/>
1819       <any namespace="##other" processContents="lax"/>
1820     </choice>
1821   </sequence>
1822 </complexType>
1823 <element name="AdviceElement" type="saml:AdviceAbstractType"/>
1824 <complexType name="AdviceAbstractType"/>
1825 <element name="Statement" type="saml:StatementAbstractType"/>
1826 <complexType name="StatementAbstractType" abstract="true"/>
1827 <element name="SubjectStatement" type="saml:SubjectStatementAbstractType"/>
1828 <complexType name="SubjectStatementAbstractType" abstract="true">
1829   <complexContent>
1830     <extension base="saml:StatementAbstractType">
1831       <sequence>
1832         <element ref="saml:Subject"/>
1833       </sequence>
1834     </extension>
1835   </complexContent>
1836 </complexType>
1837 <element name="Subject" type="saml:SubjectType"/>
1838 <complexType name="SubjectType">
1839   <choice maxOccurs="unbounded">

```

```

1840     <sequence>
1841         <element ref="saml:NameIdentifier"/>
1842         <element ref="saml:SubjectConfirmation" minOccurs="0"/>
1843     </sequence>
1844     <element ref="saml:SubjectConfirmation"/>
1845 </choice>
1846 </complexType>
1847 <element name="NameIdentifier" type="saml:NameIdentifierType"/>
1848 <complexType name="NameIdentifierType">
1849     <attribute name="SecurityDomain" type="string"/>
1850     <attribute name="Name" type="string"/>
1851 </complexType>
1852 <element name="SubjectConfirmation" type="saml:SubjectConfirmationType"/>
1853 <complexType name="SubjectConfirmationType">
1854     <sequence>
1855         <element ref="saml:ConfirmationMethod" maxOccurs="unbounded"/>
1856         <element ref="saml:SubjectConfirmationData" minOccurs="0"/>
1857         <element ref="ds:KeyInfo" minOccurs="0"/>
1858     </sequence>
1859 </complexType>
1860 <element name="SubjectConfirmationData" type="string" minOccurs="0"/>
1861 <element name="ConfirmationMethod" type="anyURI"/>
1862 <element name="AuthenticationStatement"
1863     type="saml:AuthenticationStatementType"/>
1864 <complexType name="AuthenticationStatementType">
1865     <complexContent>
1866         <extension base="saml:SubjectStatementAbstractType">
1867             <sequence>
1868                 <element ref="saml:AuthenticationLocality" minOccurs="0"/>
1869                 <element ref="saml:AuthorityBinding"
1870                 minOccurs="0" maxOccurs="unbounded"/>
1871             </sequence>
1872             <attribute name="AuthenticationMethod" type="anyURI"/>
1873             <attribute name="AuthenticationInstant" type="dateTime"/>
1874         </extension>
1875     </complexContent>
1876 </complexType>
1877 <element name="AuthenticationLocality"
1878     type="saml:AuthenticationLocalityType"/>
1879 <complexType name="AuthenticationLocalityType">
1880     <attribute name="IPAddress" type="string" use="optional"/>
1881     <attribute name="DNSAddress" type="string" use="optional"/>
1882 </complexType>
1883 <element name="AuthorityBinding" type="saml:AuthorityBindingType"/>
1884 <complexType name="AuthorityBindingType">
1885     <attribute name="AuthorityKind" type="saml:AuthorityKindType"/>
1886     <attribute name="Binding" type="anyURI"/>
1887 </complexType>
1888 <simpleType name="AuthorityKindType">
1889     <restriction base="string">
1890         <enumeration value="authentication"/>
1891         <enumeration value="attribute"/>
1892         <enumeration value="authorization"/>
1893     </restriction>
1894 </simpleType>
1895 <element name="AuthorizationDecisionStatement"
1896     type="saml:AuthorizationDecisionStatementType"/>
1897 <complexType name="AuthorizationDecisionStatementType">
1898     <complexContent>
1899         <extension base="saml:SubjectStatementAbstractType">
1900             <sequence>
1901                 <element ref="saml:Actions"/>
1902                 <element ref="saml:Evidence"/>

```

```

minOccurs="0" maxOccurs="unbounded"/>
</sequence>
<attribute name="Resource" type="anyURI" use="optional"/>
<attribute name="Decision"
type="saml:DecisionType" use="optional"/>
</extension>
</complexContent>
</complexType>
<element name="Actions" type="saml:ActionsType"/>
<complexType name="ActionsType">
<sequence>
<element ref="saml:Action" maxOccurs="unbounded"/>
</sequence>
<attribute name="Namespace" type="anyURI" use="optional"/>
</complexType>
<element name="Action" type="string"/>
<element name="Evidence" type="saml:AssertionSpecifierType"/>
<element name="AttributeStatement" type="saml:AttributeStatementType"/>
<complexType name="AttributeStatement">
<complexContent>
<extension base="saml:SubjectStatementAbstractType">
<sequence>
<element ref="saml:Attribute" maxOccurs="unbounded"/>
</sequence>
</extension>
</complexContent>
</complexType>
<element name="AttributeDesignator" type="saml:AttributeDesignatorType"/>
<complexType name="AttributeDesignatorType">
<attribute name="AttributeName" type="string"/>
<attribute name="AttributeNamespace" type="anyURI"/>
</complexType>
<element name="Attribute" type="saml:AttributeType"/>
<complexType name="AttributeType">
<complexContent>
<extension base="saml:AttributeDesignatorType">
<sequence>
<element ref="saml:AttributeValue"/>
</sequence>
</extension>
</complexContent>
</complexType>
<element name="AttributeValue" type="saml:AttributeValueType"/>
<complexType name="AttributeValueType">
<sequence>
<any namespace="##any" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
</sequence>
</complexType>
</schema>

```

8.2. Protocol Schema

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

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Phill Hallam-Baker
(VeriSign Inc.) -->
<schema
targetNamespace="http://www.oasis-open.org/committees/security/docs/draft-
sstc-schema-protocol-24.xsd"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"

```

```

1963     xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-
1964 schema-assertion-24.xsd"
1965     xmlns:samlp="http://www.oasis-open.org/committees/security/docs/draft-sstc-
1966 schema-protocol-24.xsd"
1967     xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
1968     <import
1969         namespace="http://www.oasis-open.org/committees/security/docs/draft-sstc-
1970 schema-assertion-24.xsd"
1971         schemaLocation="draft-sstc-schema-assertion-24.xsd"/>
1972     <import namespace="http://www.w3.org/2000/09/xmldsig#"
1973         schemaLocation="xmldsig-core-schema.xsd"/>
1974     <annotation>
1975         <documentation>draft-sstc-schema-protocol-24.xsd</documentation>
1976     </annotation>
1977     <simpleType name="StatusCodeType">
1978     <restriction base="string">
1979     <enumeration value="Success"/>
1980     <enumeration value="Failure"/>
1981     <enumeration value="Error"/>
1982     <enumeration value="Unknown"/>
1983     </restriction>
1984     </simpleType>
1985     <complexType name="RequestAbstractType" abstract="true">
1986         <sequence>
1987             <element ref="samlp:RespondWith"
1988                 minOccurs="0" maxOccurs="unbounded"/>
1989             <element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
1990         </sequence>
1991         <attribute name="RequestID" type="saml:IDType" use="required"/>
1992         <attribute name="MajorVersion" type="integer" use="required"/>
1993         <attribute name="MinorVersion" type="integer" use="required"/>
1994     </complexType>
1995     <element name="RespondWith" type="anyURI"/>
1996     <element name="Request" type="samlp:RequestType"/>
1997     <complexType name="RequestType">
1998         <complexContent>
1999             <extension base="samlp:RequestAbstractType">
2000                 <choice>
2001                     <element ref="samlp:Query"/>
2002                     <element ref="samlp:SubjectQuery"/>
2003                     <element ref="samlp:AuthenticationQuery"/>
2004                     <element ref="samlp:AttributeQuery"/>
2005                     <element ref="samlp:AuthorizationDecisionQuery"/>
2006                     <element ref="saml:AssertionID" maxOccurs="unbounded"/>
2007                     <element ref="samlp:AssertionArtifact" maxOccurs="unbounded"/>
2008                 </choice>
2009             </extension>
2010         </complexContent>
2011     </complexType>
2012     <element name="AssertionArtifact" type="string"/>
2013     <element name="Query" type="samlp:QueryAbstractType"/>
2014     <complexType name="QueryAbstractType" abstract="true">
2015         <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType"/>
2016     <complexType name="SubjectQueryAbstractType" abstract="true">
2017         <complexContent>
2018             <extension base="samlp:QueryAbstractType">
2019                 <sequence>
2020                     <element ref="saml:Subject"/>
2021                 </sequence>
2022             </extension>
2023         </complexContent>
2024     </complexType>
2025     <element name="AuthenticationQuery" type="samlp:AuthenticationQueryType"/>

```

```

2026 <complexType name="AuthenticationQueryType">
2027 <complexContent>
2028 <extension base="samlp:SubjectQueryAbstractType">
2029 <sequence>
2030 <element ref="saml:ConfirmationMethod" minOccurs="0"/>
2031 </sequence>
2032 </extension>
2033 </complexContent>
2034 </complexType>
2035 <element name="AttributeQuery" type="samlp:AttributeQueryType"/>
2036 <complexType name="AttributeQueryType">
2037 <complexContent>
2038 <extension base="samlp:SubjectQueryAbstractType">
2039 <sequence>
2040 <element ref="saml:AttributeDesignator"
2041 minOccurs="0" maxOccurs="unbounded"/>
2042 </sequence>
2043 </extension>
2044 </complexContent>
2045 </complexType>
2046 <element name="AuthorizationDecisionQuery"
2047 type="samlp:AuthorizationDecisionQueryType"/>
2048 <complexType name="AuthorizationDecisionQueryType">
2049 <complexContent>
2050 <extension base="samlp:SubjectQueryAbstractType">
2051 <sequence>
2052 <element ref="saml:Actions"/>
2053 <element ref="saml:Evidence"
2054 minOccurs="0" maxOccurs="unbounded"/>
2055 </sequence>
2056 <attribute name="Resource" type="anyURI"/>
2057 </extension>
2058 </complexContent>
2059 </complexType>
2060 <complexType name="ResponseAbstractType" abstract="true">
2061 <sequence>
2062 <element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
2063 </sequence>
2064 <attribute name="ResponseID" type="saml:IDType" use="required"/>
2065 <attribute name="InResponseTo" type="saml:IDType" use="required"/>
2066 <attribute name="MajorVersion" type="integer" use="required"/>
2067 <attribute name="MinorVersion" type="integer" use="required"/>
2068 </complexType>
2069 <element name="Response" type="samlp:ResponseType"/>
2070 <element name="Response" type="samlp:ResponseType"/>
2071 <complexType name="ResponseType">
2072 <complexContent>
2073 <extension base="samlp:ResponseAbstractType">
2074 <sequence>
2075 <element ref="samlp:Status"/>
2076 <element ref="saml:Assertion"
2077 minOccurs="0" maxOccurs="unbounded"/>
2078 </sequence>
2079 </extension> <extension base="samlp:ResponseAbstractType">
2080 <sequence>
2081 <element ref="samlp:StatusReason"
2082 minOccurs="0" maxOccurs="unbounded"/>
2083 <element ref="saml:Assertion"
2084 minOccurs="0" maxOccurs="unbounded"/>
2085 </sequence>
2086 <attribute name="StatusCode"
2087 type="samlp:StatusCodeType" use="required"/>
2088 </extension>

```

```

2089     </complexContent>
2090 </complexType>
2091 <element name="Status" type="samlp:StatusType"/>
2092 <complexType name="StatusType">
2093   <sequence>
2094     <element ref="samlp:StatusCode"/>
2095     <element ref="samlp:StatusMessage"
2096       minOccurs="0" maxOccurs="unbounded"/>
2097     <element ref="samlp:StatusDetail" minOccurs="0"/>
2098   </sequence>
2099 </complexType>
2100 <element name="StatusCode" type="samlp:StatusCodeType"/>
2101 <complexType name="StatusCodeType">
2102   <attribute name="Value" type="samlp:StatusCodeEnumType"/>
2103   <attribute name="SubStatusCode" type="QName" use="optional"/>
2104 </complexType>
2105 <element name="StatusMessage" type="string"/>
2106 <simpleType name="StatusCodeEnumType">
2107   <restriction base="QName">
2108     <enumeration value="samlp:Success"/>
2109     <enumeration value="samlp:VersionMismatch"/>
2110     <enumeration value="samlp:Receiver"/>
2111     <enumeration value="samlp:Sender"/>
2112   </restriction>
2113 </simpleType>
2114 <element name="StatusDetail" type="samlp:StatusDetailType"/>
2115 <complexType name="StatusDetailType">
2116   <sequence>
2117     <any namespace="##any"
2118       processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2119   </sequence>
2120 </complexType> <del><element name="StatusReason" type="string"/></del>
2121 </schema>
2122

```

9. References

- [Needham78Kerberos]** R. Needham et al., *Using Encryption for Authentication in Large Networks of Computers*, Communications of the ACM, Vol. 21 (12), pp. 993-999, December 1978.
- [Kern-84]** B. Kernighan, Rob Pike *The UNIX Programming Environment*, (March 1984) Prentice Hall Computer Books;
- [PKCS1]** B. Kaliski, *PKCS #1: RSA Encryption Version 2.0*, RSA Laboratories, also IETF RFC 2437, October 1998. <http://www.ietf.org/rfc/rfc2437.txt>
- [PKCS7]** B. Kaliski., "PKCS #7: Cryptographic Message Syntax, Version 1.5.", RFC 2315, March 1998.
- [RFC 1510]** J. Kohl, C. Neuman. *The Kerberos Network Authentication Service (V5)*. September 1993. <http://www.ietf.org/rfc/rfc1510.txt>
- [RFC 2246]** T. Dierks, C. Allen. *The TLS Protocol Version 1.0*. January 1999. <http://www.ietf.org/rfc/rfc2246.txt>
- [RFC 2630]** R. Housley. *Cryptographic Message Syntax*. June 1999. <http://www.ietf.org/rfc/rfc630.txt>
- [RFC 2648]** R. Moats. *A URN Namespace for IETF Documents*. August 1999. <http://www.ietf.org/rfc/rfc2648.txt>
- [RFC 3075]** D. Eastlake, J. Reagle, D. Solo. *XML-Signature Syntax and Processing*. March 2001. <http://www.ietf.org/rfc/rfc3075.txt>
- [RFC2104]** H. Krawczyk et al., *HMAC: Keyed Hashing for Message Authentication*, <http://www.ietf.org/rfc/rfc2104.txt>, IETF RFC 2104, February 1997.
- [RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997
- [SAMLBind]** P. Mishra et al., *Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML)*, <http://www.oasis-open.org/committees/security/docs/draft-sstc-bindings-model-07.pdf>, OASIS, December 2001.
- [SAMLConform]** **TBS**
- [SAMLGloss]** J. Hodges et al., *Glossary for the OASIS Security Assertion Markup Language (SAML)*, <http://www.oasis-open.org/committees/security/docs/draft-sstc-glossary-02.pdf>, OASIS, December 2001.
- [SAML-XSD]** P. Hallam-Baker et al., *SAML protocol schema*, <http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-protocol-21.xsd>, OASIS, December 2001.
- [SAMLSecure]** **TBS**
- [SAML-XSD]** P. Hallam-Baker et al., *SAML assertion schema*, <http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-21.xsd>, OASIS, December 2001.
- [Schema1]** H. S. Thompson et al., *XML Schema Part 1: Structures*, <http://www.w3.org/TR/xmlschema-1/>, World Wide Web Consortium Recommendation, May 2001.
- [Schema2]** P. V. Biron et al., *XML Schema Part 2: Datatypes*, <http://www.w3.org/TR/xmlschema-2>, World Wide Web Consortium Recommendation, May 2001.
- [XMLEnc]** *XML Encryption Specification*, In development.

2170	[XMLSig]	D. Eastlake et al., <i>XML-Signature Syntax and Processing</i> ,
2171		http://www.w3.org/TR/xmldsig-core/ , World Wide Web Consortium.
2172	[XMLSig-XSD]	XML Signature Schema available from http://www.w3.org/TR/2000/CR-
2173		xmldsig-core-20001031/xmldsig-core-schema.xsd
2174	[XTAML]	P. Hallam-Baker, <i>XML Trust Axiom Markup Language 1.0</i> ,
2175		http://www.xmltrustcenter.org/ , VeriSign Inc. September 2001.

Appendix A. Notices

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

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

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

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

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

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