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

Document identifier: draft-sstc-core-26

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

Publication date: January 10th 2002

Maturity Level: Committee Working Draft

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

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

Editors:

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

Contributors:

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

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 **[SAMLP-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 onlybe 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-26.xsd
```

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

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

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

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

```
http://www.w3.org/2000/09/xmldsig#
```

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

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

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

1.2.2. Comparing SAML values

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

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

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

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

1.3. SAML Concepts (Non-Normative)

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

1.3.1. Overview

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

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

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

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

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

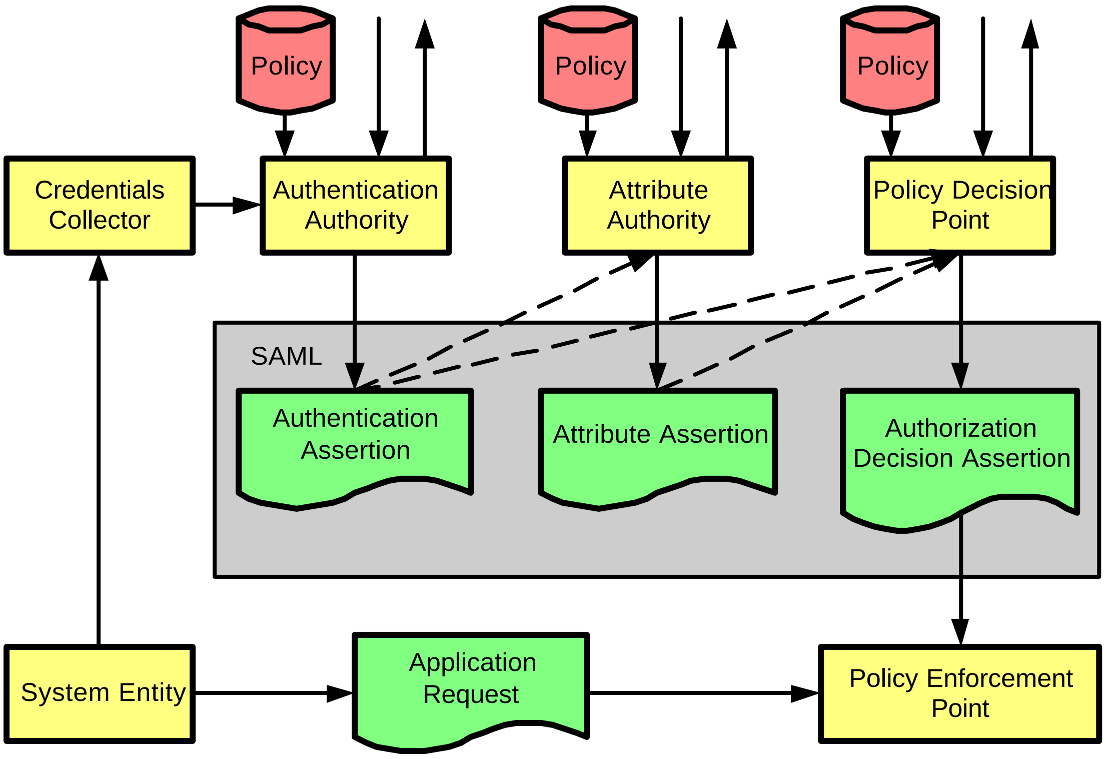

Figure 1 The SAML Domain Model

One major design ~~center~~ goal 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 separatespecification **[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-25/rwedc**~~**http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#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.

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

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

1.3.3. SAML and Extensibility

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

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

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 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 contain~~s~~ the specifics, while an outer generic assertion element provides information that is common to all of 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-26.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-
schema-assertion-26.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-26.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 is 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.~~

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

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

Values 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 accidentally 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}. This requirement MAY be met by applying Base64 encoding to a randomly chosen value 128 or 160 bits in length.

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** and **IDReferenceType** simple types:

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

`<AssertionIDReference>`

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:AssertionIDReference"/>
            <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 of the assertion's `AssertionID` attribute.

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

```
    <element ~~name="AssertionID" type="saml:IDType"/>~~name="AssertionIDReference"
type="saml:IDReferenceType"/>
```

2.3.3. Element <Assertion>

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

`MajorVersion` [Required]

The major version of this assertion. The identifier for the version of SAML defined in this specification is `1`. Processing of this attribute is specified in Section 1.1.1.

`MinorVersion` [Required]

The minor version of this assertion. The identifier for the version of SAML defined in this specification is `0`. Processing of this attribute is specified in Section 1.1.1.

`AssertionID` [Required]

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

`Issuer` [Required]

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

`IssueInstant` [Required]

The time instant of ~~issue. It has the type **dateTime**, which is built in to the W3C XML Schema Datatypes specification **[Schema2]**~~issue in UTC as described in section 1.2.1.

`<Conditions>` [Optional]

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

`<Advice>` [Optional]

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

`<Signature>` [Optional]

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

One or more of the following statement elements:

`<Statement>`

A statement defined in an extension schema.

`<SubjectStatement>`

A subject statement defined in an extension schema.

`<AuthenticationStatement>`

An authentication statement.

`<AuthorizationDecisionStatement>`

An authorization decision statement.

`<AttributeStatement>`

An attribute statement.

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

```
    <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”/>~~ref="ds:Signature" minOccurs="0"/>
        </sequence>
        <attribute name="MajorVersion" type="integer" use="required"/>
        <attribute name="MinorVersion" type="integer" use="required"/>
        <attribute name="AssertionID" type="saml:IDType" use="required"/>
        <attribute name="Issuer" type="string" use="required"/>
        <attribute name="IssueInstant" type="dateTime" use="required"/>
    </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 status 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`.

Note that an assertion that has validity stat us ‘Valid’ may not be trustworthy by reasons such as not being issued by a trustworthy issuer or not being authenticated by a trustworthy signature.

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~~MAY contain the following elements and attributes:

`NotBefore` [Optional]

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

`NotOnOrAfter` [Optional]

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

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

[Any Number]

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:

```
<element name="Conditions" type="saml:ConditionsType"/>
<complexType name="ConditionsType">
   <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="saml:Condition"/>
      <element ref="saml:AudienceRestrictionCondition"/>
      <element ref="saml:TargetRestrictionCondition"/>
   </choice>
   <attribute name="NotBefore" type="dateTime" use="optional"/>
   <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>
</complexType>
```

2.3.3.1.1 Attributes NotBefore and NotOnOrAfter

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

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

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

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

2.3.3.1.2 Element <Condition>

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

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

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

2.3.3.1.3 Elements <AudienceRestrictionCondition> and <Audience>

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

`<Audience>`

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

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

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

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

```
    <element name="AudienceRestrictionCondition"
          type="saml:AudienceRestrictionConditionType"/>
    <complexType name="AudienceRestrictionConditionType">
        <complexContent>
            <extension base="saml:ConditionAbstractType">
                <sequence>
                    <element ref="saml:Audience"
                        ~~minOccurs="1"~~ maxOccurs="unbounded"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <element name="Audience" type="anyURI"/>
```

2.3.3.1.4 ~~Condition Type TargetRestrictionType~~ Elements <TargetRestrictionCondition> and <Target>

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

`<Target>`
: A URI that identifies an intended relying party.

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

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

```
<element name="TargetRestrictionCondition"
         type="saml:TargetRestrictionConditionType"/>
<complexType name="TargetRestrictionConditionType">
    <complexContent>
        <extension base="saml:ConditionAbstractType">
            <sequence>
                <element ref="saml:Target"
                         minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<element name="Target" type="anyURI"/>
```

2.3.3.2. Elements <Advice> and <AdviceElement>

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

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

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

?? Include evidence supporting the assertion claims to be cited, either directly (through incorporating the claims) or indirectly (by reference to the supporting assertions).

?? State a proof of the assertion claims.

?? Specify the timing and distribution points for updates to the assertion.

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

```
<element name="Advice" type="saml:AdviceType"/>
<complexType name="AdviceType">
    <sequence>
        <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="saml:AssertionSpecifier"/>
            <element ref="saml:AdviceElement"/>
            <any namespace="##other" processContents="lax"/>
        </choice>
    </sequence>
</complexType>
<element name="AdviceElement" type="saml:AdviceAbstractType"/>
```

```
    <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:

```
    <element name="Statement" type="saml:StatementAbstractType"/>
    <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:

```
    <element name="SubjectStatement" type="saml:SubjectStatementAbstractType"/>
    <complexType name="SubjectStatementAbstractType" abstract="true">
        <complexContent>
            <extension base="saml:StatementAbstractType">
                <sequence>
                    <element ref="saml:Subject"/>
                </sequence>
            </extension>
        </complexContent>
    </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.

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

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

```
        <choice~~ maxOccurs="unbounded"~~>
            <sequence>
                <element ref="saml:NameIdentifier"/>
                <element ref="saml:SubjectConfirmation" minOccurs="0"/>
            </sequence>
            <element ref="saml:SubjectConfirmation"/>
        </choice>
    </complexType>
```

2.4.2.2. Element <NameIdentifier>

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

`SecurityDomain`
: The security domain governing the name of the subject.

`Name`
: The name of the subject.

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

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

```
    <element name="NameIdentifier" type="saml:NameIdentifierType"/>
    <complexType name="NameIdentifierType">
        <attribute name="SecurityDomain" type="string"/>
        <attribute name="Name" type="string"/>
    </complexType>
```

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

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

`<ConfirmationMethod>` [One or more]
: A URI that identifies a protocol to be used to authenticate the subject. URIs identifying common authentication protocols are listed in Section 7.

`<SubjectConfirmationData>` ~~[Zero or more]~~
: [Optional]
: Additional authentication information to be used by a specific authentication protocol.

`<ds:KeyInfo>` [Optional]
: An XML Signature **[XMLSig]** element that specifies a cryptographic key held by the subject.

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

```
    <element name="SubjectConfirmation" type="saml:SubjectConfirmationType"/>
    <complexType name="SubjectConfirmationType">
        <sequence>
            <element ref="saml:ConfirmationMethod" maxOccurs="unbounded"/>
            <element ref="saml:SubjectConfirmationData" minOccurs="0"/>
            <element ref="ds:KeyInfo" minOccurs="0"/>
        </sequence>
    </complexType>
    <element name="SubjectConfirmationData" type="string"~~ minOccurs="0"~~/>
    <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]~~
: [Optional]
: A URI that specifies the type of authentication that took place. URIs identifying common authentication protocols are listed in Section 7.

`AuthenticationInstant` ~~[Required]~~
: [Optional]
: Specifies the time at which the authentication took place. The time value is encoded in UTC as described in section 1.2.1.

`<AuthenticationLocality>` [Optional]
: Specifies the DNS domain name and IP address for the system entity ~~that~~from which the Subject was apparently authenticated.

`<AuthenticationBinding>` [Any Number]
: Indicates that additional information about the subject of the statement may be available.

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

```
    <element name="AuthenticationStatement"
             type="saml:AuthenticationStatementType"/>
    <complexType name="AuthenticationStatementType">
        <complexContent>
            <extension base="saml:SubjectStatementAbstractType">
                <sequence>
                    <element ref="saml:AuthenticationLocality" minOccurs="0"/>
                    <element ref="saml:AuthorityBinding"
                             minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
                <attribute name="AuthenticationMethod" type="anyURI"/>
                <attribute name="AuthenticationInstant" type="dateTime"/>
            </extension>
        </complexContent>
    </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]~~
: [Optional]
: 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:

```
    <element name="AuthenticationLocality"
             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,~~ may be used to indicate to a relying party receiving an AuthenticationStatement that a SAML authority may be available to provide additional information about the subject of the statement. A single SAML authority may advertise its presence over multiple protocol bindings, at multiple locations, and as more than one kind of ~~authorization) that performed the authenticationand points to it via URI.~~

~~`AuthorityKind` [Optional]~~

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

~~`Binding` [Optional]~~

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

`AuthorityKind` [Required]

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

`Location` [Required]

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

`Binding` [Required]

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

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="Location" type="anyURI" use="required"/>
        <attribute name="Binding" type="anyURI" use="required"/>
    </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]~~

[Any Number]

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-26#rwedc-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.~~ requestor and the Authorization Authority. For example, in the case that the ~~client~~requestor presented an assertion to the ~~service~~Authorization Authority in a request, the ~~service~~Authorization Authority MAY use that assertion as evidence in making its response without endorsing the assertion as valid either to the ~~client~~requestor 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"/>
    <complexType name="AttributeStatementType">
        <complexContent>
            <extension base="saml:SubjectStatementAbstractType">
                <sequence>
                    <element ref="saml:Attribute" maxOccurs="unbounded"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
```

2.4.5.1. Elements <AttributeDesignator> and <Attribute>

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

AttributeNamespace ~~[Required]~~
    [Optional]
    The namespace in which the AttributeName elements are interpreted.

AttributeName ~~[Required]~~
    [Optional]
    The name of the attribute.

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

```
    <element name="AttributeDesignator" type="saml:AttributeDesignatorType"/>
    <complexType name="AttributeDesignatorType">
        <attribute name="AttributeName" type="string"/>
        <attribute name="AttributeNamespace" type="anyURI"/>
    </complexType>
```

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

<AttributeValue> ~~[Required]~~

[Any Number]

The value of the attribute.

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

```
    <element name="Attribute" type="saml:AttributeType"/>
    <complexType name="AttributeType">
        <complexContent>
            <extension base="saml:AttributeDesignatorType">
                <sequence>
                    <element ref="saml:AttributeValue" maxOccurs="unbounded"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
```

2.4.5.1.1 Element <AttributeValue>

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

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

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

```
    <element name="AttributeValue"
type="saml:AttributeValueType"/>type="anyType"/>
~~    <complexType name="AttributeValueType">~~
~~        <sequence>~~
~~            <any namespace="##any" processContents="lax"~~
~~                minOccurs="0" maxOccurs="unbounded"/>~~
~~        </sequence>~~
~~    </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~~requestors MAY in addition use the SAML request-response protocol defined by the `<Request>` and `<Response>` elements. The ~~client~~ requestor sends a `<Request>` element to a SAML ~~service, and the service~~authority, and the authority generates a `<Response>` element, as shown in Figure 2~~Figure 2~~.

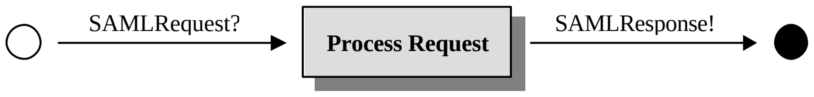

Figure 2: SAML Request-Response Protocol

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-26.xsd"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:samlp="http://www.oasis-open.org/committees/security/docs/draft-sstc-
schema-protocol-26.xsd"
    xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-
schema-assertion-26.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    elementFormDefault="unqualified">
    <import namespace="http://www.oasis-open.org/committees/security/docs/draft-
sstc-schema-assertion-26.xsd"
        schemaLocation="draft-sstc-schema-assertion-26.xsd"/>
    <import namespace="http://www.w3.org/2000/09/xmldsig#"
        schemaLocation="xmldsig-core-schema.xsd"/>
    <annotation>
        <documentation>draft-sstc-schema-protocol-26.xsd</documentation>
    </annotation>
…
</schema>
```

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 and elements 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 3.4.2.

`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 3.4.2.

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

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

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

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

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

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.`

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

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

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

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

`SingleStatement`
An assertion carrying exactly one statement element.

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

`AuthenticationStatement`
: An assertion carrying an Authentication statement.

`AuthorizationDecisionStatement`
: An assertion carrying an Authorization Decision statement.

`AttributeStatement`
: An assertion carrying an Attribute statement.

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

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

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

3.2.2. Element <Request>

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

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

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

`<AuthenticationQuery>`
: Makes a query for authentication information.

`<AttributeQuery>`
: Makes a query for attribute information.

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

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

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

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

```
    <element name="Request" type="samlp:RequestType"/>
    <complexType name="RequestType">
        <complexContent>
            <extension base="samlp:RequestAbstractType">
                <choice>
                    <element ref="samlp:Query"/>
                    <element ref="samlp:SubjectQuery"/>
                    <element ref="samlp:AuthenticationQuery"/>
                    <element ref="samlp:AttributeQuery"/>
                    <element ref="samlp:AuthorizationDecisionQuery"/>
                    <element ref="saml:AssertionIDReference" maxOccurs="unbounded"/>
                    <element ref="samlp:AssertionArtifact" maxOccurs="unbounded"/>
                </choice>
            </extension>
        </complexContent>
    </complexType>
```

3.2.3. Element <AssertionArtifact>

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

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

```
    <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:

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

```
    <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType"/>
    <complexType name="SubjectQueryAbstractType" abstract="true">
        <complexContent>
            <extension base="samlp:QueryAbstractType">
                <sequence>
                    <element ref="saml:Subject"/>
                </sequence>
            </extension>
        </complexContent>
    </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~~assertions containing~~an~~ authentication statements. 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.” TheA successful response will be in the form of an assertion containing an attribute ~~statement.~~assertions containing attribute statements. This element is of type **AttributeQueryType**, which extends **SubjectQueryAbstractType** with the addition of the following element and attribute:

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

Each `<AttributeDesignator>` element specifies an attribute whose value is to be returned. If no attributes are specified, 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>
        </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~~A successful response will be in the form of ~~an assertion~~assertions containing~~an~~ authorization decision statements. 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]~~

[Any Number]

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:

```
    <element name="AuthorizationDecisionQuery"
type="samlp:AuthorizationDecisionQueryType"/>
    <complexType name="AuthorizationDecisionQueryType">
        <complexContent>
            <extension base="samlp:SubjectQueryAbstractType">
                <sequence>
                    <element ref="saml:Actions"/>
                    <element ref="saml:Evidence"
                            minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
                <attribute name="Resource" type="anyURI" use="required"/>
            </extension>
        </complexContent>
    </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 and elements 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 1.1.1.

`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 1.1.1.

`IssueInstant` [Optional]

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

`<Signature>` [Any Number]

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

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

```
    <complexType name="ResponseAbstractType" abstract="true">
        <sequence>
            <element ref = ~~“ds:Signature” minOccurs=”0”
maxOccurs="unbounded"/>~~"ds:Signature" minOccurs="0"/>
        </sequence>
        <attribute name="ResponseID" type="saml:IDType" use="required"/>
        <attribute name="InResponseTo" ~~type="saml:IDType" use="required"/>~~
type="saml:IDReferenceType"
           use="required"/>
        <attribute name="MajorVersion" type="integer" use="required"/>
        <attribute name="MinorVersion" type="integer" use="required"/>
        <attribute name="IssueInstant" type="dateTime" use="required"/>
    </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>` [Required] (see Section 3.4.3)
: A code representing the status of the corresponding request.

`<Assertion>` [Any Number] (see Section 2.3.3)
: Specifies an assertion 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>
        </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:

`Value` [Required]
: The status code value as defined below.

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

The following **StatusCode** values are defined:

`Success`
: The request succeeded.

`VersionMismatch`
: The receiver could not process the request because the version was incorrect.

~~`Reciever`~~
`Receiver`
: The request could not be performed due to an error at the receiving end.

`Sender`
: The request could not be performed due to an error in the sender or in the request

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

```
    <element name="StatusCode" type="samlp:StatusCodeType"/>
    <complexType name="StatusCodeType">
        <sequence>
            <element ref="samlp:SubStatusCode" minOccurs="0"/>
        </sequence>
        <attribute name="Value" type="samlp:StatusCodeEnumType" use="required"/>
    </complexType>
    <simpleType name="StatusCodeEnumType">
        <restriction base="QName">
            <enumeration value="samlp:Success"/>
            <enumeration value="samlp:VersionMismatch"/>
            <enumeration value="samlp:Receiver"/>
            <enumeration value="samlp:Sender"/>
        </restriction>
    </simpleType>
```

3.4.3.2. Element <SubStatusCode>

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

`Value` [Required]
: The status code value as defined below.

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

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

`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 `<SubStatusCode>` element and its **SubStatusCodeType** complex type:

```
   <element name="SubStatusCode" type="samlp:SubStatusCodeType"/>
   <complexType name="SubStatusCodeType">
      <sequence>
         <element ref="samlp:SubStatusCode" minOccurs="0"/>
      </sequence>
      <attribute name="Value" type="QName"  use="required"/>
   </complexType>
```

3.4.3.3. Element <StatusMessage>

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

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

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

3.4.3.4. 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>
```

~~3.4.4. 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"/>~~
~~            <enumeration value="Unknown"/>~~
~~        </restriction>~~
```

3.4.4. ~~</simpleType>~~Responses to <AuthenticationQuery> and <AttributeQuery>

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

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

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

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

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:

$Major_B > Major_A$? (($Major_B = Major_A$) ? $Minor_B$ ~~=~~$\geq$ $Minor_A$)

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:** ($Major_B = Major_A$) ? ($Minor_B$ ~~=~~$\geq$ $Minor_A$)
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

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

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

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

`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.

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

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

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

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

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

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

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

5.1. Signing Assertions

All SAML assertions MAY be signed using the XML Signature. This is reflected in the assertion schema – Section ~~2.3.3.~~2.3.

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.~~ 3.2 & 3.4.

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.

A SAML request or response may be viewed as inheriting a signature from a super signature, if the super signature applies to all of 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~~5.1

5.4. XML Signature Profile

The XML Signature **[XMLSig]** 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 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.~~ assertions and protocols. SAML processors should support use of RSA signing and verification for public key operations.

5.4.2. CanonicalizationMethod

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

5.4.3. Transforms

~~[Sig]~~**[XMLSig]** 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]~~**[XMLSig]** 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~~through 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:

?? `<AuthenticationStatement>`

?? `<AuthorizationDecisionStatement>`

?? `<AttributeStatement>`

?? `<AudienceRestrictionCondition>`

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

?? `<AttributeValue>`

?? `<Advice>`

6.2. Protocol Schema Extension

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

?? `<Query>`

?? `<SubjectQuery>`

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

?? `<Request>`

?? `<AuthenticationQuery>`

?? `<AuthorizationDecisionQuery>`

?? `<AttributeQuery>`

?? `<Response>`

6.3. Use of Type Derivation and Substitution Groups

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

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

```
<saml:Assertion …>
  <saml:Statement xsi:type="new:NewStatementType">
  …
  </saml:Statement>
</saml:Assertion>
```

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

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

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

```
<saml:Assertion …>
   <new:NewStatement>
     …
   </new:NewStatement>
</saml:Assertion>
```

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

The advantages of type derivation are as follows:

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

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

The advantage of substitution groups is that a document can be explained without the need to 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-~~25/~~26
```

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-25/artifact~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#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-25/artifact-sha1~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#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-25/Holder-Of-Key~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#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-25/sender-vouches~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#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-25/password~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#password

`<SubjectConfirmationData>`: *Base64* (*Password*)

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

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

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

URI: ~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/password-sha1~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#password-sha1

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

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

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

7.1.7. Kerberos

URI: urn:ietf:rfc:1510

`<SubjectConfirmationData>`: A Kerberos Ticket

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

7.1.8. SSL/TLS Certificate Based Client Authentication:

URI: urn:ietf:rfc:2246

`<ds:KeyInfo>`: Any cryptographic key

7.1.9. Object Authenticator (SHA-1):

URI: ~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/object-sha1~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#object-sha1

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

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

7.1.10. PKCS#7

URI: urn:ietf:rfc:2315

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

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

7.1.11. Cryptographic Message Syntax

URI: urn:ietf:rfc:2630

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

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

7.1.12. XML Digital Signature

URI: ~~urn:ietf:rfc:2630~~urn:ietf:rfc:3075

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

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

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

7.2. Action Namespace Identifiers

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

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

URI: ~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/rwedc~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#rwedc

Defined actions:

`Read Write Execute Delete Control`

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

`Read`
The subject may read the resource

`Write`
The subject may modify the resource

`Execute`
The subject may execute the resource

`Delete`
The subject may delete the resource

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

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

URI: ~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-25/rwedc-negation~~http://www.oasis-open.org/committees/security/docs/draft-sstc-core-26#rwedc-negation

Defined actions:

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

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

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

7.2.3. Get/Head/Put/Post:

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

Defined actions:

`GET HEAD PUT POST`

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

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

7.2.4. UNIX File Permissions:

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

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

The action string is a four digit numeric code:

extended user group world

Where the *extended* access permission has the value

+2 if sgid is set

+4 if suid is set

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

+1 if execute permission is granted

+2 if write permission is granted

+4 if read permission is granted

For example `0754` denotes the UNIX file access permission: user read, write and execute, group 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-26.xsd"
   xmlns="http://www.w3.org/2001/XMLSchema" xmlns:saml="http://www.oasis-
open.org/committees/security/docs/draft-sstc-schema-assertion-26.xsd"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
elementFormDefault="unqualified">
    <import namespace="http://www.w3.org/2000/09/xmldsig#"
            schemaLocation="xmldsig-core-schema.xsd"/>
    <annotation>
        <documentation>draft-sstc-schema-assertion-26.xsd</documentation>
    </annotation>
    <simpleType name="IDType">
        <restriction base="string"/>
    </simpleType>
    <simpleType name="IDReferenceType">
        <restriction base="string"/>
    </simpleType>
    <simpleType name="DecisionType">
        <restriction base="string">
            <enumeration value="Permit"/>
            <enumeration value="Deny"/>
            <enumeration value="Indeterminate"/>
        </restriction>
    </simpleType>
    <element name="AssertionSpecifier" type="saml:AssertionSpecifierType"/>
    <complexType name="AssertionSpecifierType">
        <choice>
            <element ref="saml:AssertionIDReference"/>
            <element ref="saml:Assertion"/>
        </choice>
    </complexType>
    <element ~~name="AssertionID" type="saml:IDType"/>~~name="AssertionIDReference"
type="saml:IDReferenceType"/>
    <element name="Assertion" type="saml:AssertionType"/>
    <complexType name="AssertionType">
        <sequence>
            <element ref="saml:Conditions" minOccurs="0"/>
            <element ref="saml:Advice" minOccurs="0"/>
            <choice~~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"/>~~"ds:Signature" minOccurs="0"/>
        </sequence>
```

```
      <attribute name="MajorVersion" type="integer" use="required"/>
      <attribute name="MinorVersion" type="integer" use="required"/>
      <attribute name="AssertionID" type="saml:IDType" use="required"/>
      <attribute name="Issuer" type="string" use="required"/>
      <attribute name="IssueInstant" type="dateTime" use="required"/>
   </complexType>
   <element name="Conditions" type="saml:ConditionsType"/>
   <complexType name="ConditionsType">
      <choice minOccurs="0" maxOccurs="unbounded">
         <element ref="saml:Condition"/>
         <element ref="saml:AudienceRestrictionCondition"/>
      </choice>
      <attribute name="NotBefore" type="dateTime" use="optional"/>
      <attribute name="NotOnOrAfter" type="dateTime" use="optional"/>
   </complexType>
   <element name="Condition" type="saml:ConditionAbstractType"/>
   <complexType name="ConditionAbstractType" abstract="true"/>
   <element name="AudienceRestrictionCondition"
         type="saml:AudienceRestrictionConditionType"/>
   <complexType name="AudienceRestrictionConditionType">
      <complexContent>
         <extension base="saml:ConditionAbstractType">
            <sequence>
               <element ref="saml:Audience" maxOccurs="unbounded"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <element name="Audience" type="anyURI"/>
   <element name="TargetRestrictionCondition"
         type="saml:TargetRestrictionConditionType"/>
   <complexType name="TargetRestrictionConditionType">
      <complexContent>
         <extension base="saml:ConditionAbstractType">
            <sequence>
               <element ref="saml:Target"
                        minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <element name="Target" type="anyURI"/>
   <element name="Advice" type="saml:AdviceType"/>
   <complexType name="AdviceType">
      <sequence>
         <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="saml:AssertionSpecifier"/>
            <element ref="saml:AdviceElement"/>
            <any namespace="##other" processContents="lax"/>
         </choice>
      </sequence>
   </complexType>
   <element name="AdviceElement" type="saml:AdviceAbstractType"/>
   <complexType name="AdviceAbstractType"/>
   <element name="Statement" type="saml:StatementAbstractType"/>
   <complexType name="StatementAbstractType" abstract="true"/>
   <element name="SubjectStatement" type="saml:SubjectStatementAbstractType"/>
   <complexType name="SubjectStatementAbstractType" abstract="true">
      <complexContent>
         <extension base="saml:StatementAbstractType">
            <sequence>
               <element ref="saml:Subject"/>
            </sequence>
            </extension>
        </complexContent>
    </complexType>
    <element name="Subject" type="saml:SubjectType"/>
    <complexType name="SubjectType">
        <choice~~ maxOccurs="unbounded"~~>
            <sequence>
                <element ref="saml:NameIdentifier"/>
                <element ref="saml:SubjectConfirmation" minOccurs="0"/>
            </sequence>
            <element ref="saml:SubjectConfirmation"/>
        </choice>
    </complexType>
    <element name="NameIdentifier" type="saml:NameIdentifierType"/>
    <complexType name="NameIdentifierType">
        <attribute name="SecurityDomain" type="string"/>
        <attribute name="Name" type="string"/>
    </complexType>
    <element name="SubjectConfirmation" type="saml:SubjectConfirmationType"/>
    <complexType name="SubjectConfirmationType">
        <sequence>
            <element ref="saml:ConfirmationMethod" maxOccurs="unbounded"/>
            <element ref="saml:SubjectConfirmationData" minOccurs="0"/>
            <element ref="ds:KeyInfo" minOccurs="0"/>
        </sequence>
    </complexType>
    <element name="SubjectConfirmationData" type="string"~~ minOccurs="0"~~/>
    <element name="ConfirmationMethod" type="anyURI"/>
    <element name="AuthenticationStatement"
             type="saml:AuthenticationStatementType"/>
    <complexType name="AuthenticationStatementType">
        <complexContent>
            <extension base="saml:SubjectStatementAbstractType">
                <sequence>
                    <element ref="saml:AuthenticationLocality" minOccurs="0"/>
                    <element ref="saml:AuthorityBinding"
                             minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
                <attribute name="AuthenticationMethod" type="anyURI"/>
                <attribute name="AuthenticationInstant" type="dateTime"/>
            </extension>
        </complexContent>
    </complexType>
    <element name="AuthenticationLocality"
             type="saml:AuthenticationLocalityType"/>
    <complexType name="AuthenticationLocalityType">
        <attribute name="IPAddress" type="string" use="optional"/>
        <attribute name="DNSAddress" type="string" use="optional"/>
    </complexType>
    <element name="AuthorityBinding" type="saml:AuthorityBindingType"/>
    <complexType name="AuthorityBindingType">
        <attribute name="AuthorityKind" type="saml:AuthorityKindType"/>
        <attribute name="Location" type="anyURI" use="required"/>
        <attribute name="Binding" type="anyURI" use="required"/>
    </complexType>
    <simpleType name="AuthorityKindType">
        <restriction base="string">
            <enumeration value="authentication"/>
            <enumeration value="attribute"/>
            <enumeration value="authorization"/>
        </restriction>
    </simpleType>
    <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>
    <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="AttributeStatementType">
        <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" maxOccurs="unbounded"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <element name="AttributeValue"
~~type="saml:AttributeValueType"/>~~type="saml:anyType"/>
~~    <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 **[SAMLP-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-26.xsd"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-
schema-assertion-26.xsd"
   xmlns:samlp="http://www.oasis-open.org/committees/security/docs/draft-sstc-
schema-protocol-26.xsd"
   xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
   <import
       namespace="http://www.oasis-open.org/committees/security/docs/draft-sstc-
schema-assertion-26.xsd"
       schemaLocation="draft-sstc-schema-assertion-26.xsd"/>
   <import namespace="http://www.w3.org/2000/09/xmldsig#"
          schemaLocation="xmldsig-core-schema.xsd"/>
   <annotation>
      <documentation>draft-sstc-schema-protocol-26.xsd</documentation>
   </annotation>
   <complexType name="RequestAbstractType" abstract="true">
      <sequence>
         <element ref="samlp:RespondWith"
                  minOccurs="0" maxOccurs="unbounded"/>
         <element ref = ~~"ds:Signature" minOccurs="0"
maxOccurs="unbounded"/>~~"ds:Signature" minOccurs="0"/>
      </sequence>
      <attribute name="RequestID" type="saml:IDType" use="required"/>
      <attribute name="MajorVersion" type="integer" use="required"/>
      <attribute name="MinorVersion" type="integer" use="required"/>
      <attribute name="IssueInstant" type="dateTime" use="required"/>
   </complexType>
   <element name="RespondWith" type="anyURI"/>
   <element name="Request" type="samlp:RequestType"/>
   <complexType name="RequestType">
      <complexContent>
         <extension base="samlp:RequestAbstractType">
            <choice>
               <element ref="samlp:Query"/>
               <element ref="samlp:SubjectQuery"/>
               <element ref="samlp:AuthenticationQuery"/>
               <element ref="samlp:AttributeQuery"/>
               <element ref="samlp:AuthorizationDecisionQuery"/>
               <element ref="saml:AssertionID" maxOccurs="unbounded"/>
               <element ref="samlp:AssertionArtifact" maxOccurs="unbounded"/>
            </choice>
         </extension>
      </complexContent>
   </complexType>
   <element name="AssertionArtifact" type="string"/>
   <element name="Query" type="samlp:QueryAbstractType"/>
   <complexType name="QueryAbstractType" abstract="true"/>
   <element name="SubjectQuery" type="samlp:SubjectQueryAbstractType"/>
   <complexType name="SubjectQueryAbstractType" abstract="true">
      <complexContent>
         <extension base="samlp:QueryAbstractType">
            <sequence>
```

```
                    <element ref="saml:Subject"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <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>
    <element name="AttributeQuery" type="samlp:AttributeQueryType"/>
    <complexType name="AttributeQueryType">
        <complexContent>
            <extension base="samlp:SubjectQueryAbstractType">
                <sequence>
                    <element ref="saml:AttributeDesignator"
                               minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <element name="AuthorizationDecisionQuery"
               type="samlp:AuthorizationDecisionQueryType"/>
    <complexType name="AuthorizationDecisionQueryType">
        <complexContent>
            <extension base="samlp:SubjectQueryAbstractType">
                <sequence>
                    <element ref="saml:Actions"/>
                    <element ref="saml:Evidence"
                               minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
                <attribute name="Resource" type="anyURI" use="required"/>
            </extension>
        </complexContent>
    </complexType>
    <complexType name="ResponseAbstractType" abstract="true">
        <sequence>
            <element ref = ~~“ds:Signature” minOccurs=”0”~~
~~maxOccurs="unbounded"/>~~"ds:Signature" minOccurs="0"/>
        </sequence>
        <attribute name="ResponseID" type="saml:IDType" use="required"/>
        <attribute name="InResponseTo" ~~type="saml:IDType" use="required"/>~~
type="saml:IDReferenceType"
            use="required"/>
        <attribute name="MajorVersion" type="integer" use="required"/>
        <attribute name="MinorVersion" type="integer" use="required"/>
        <attribute name="IssueInstant" type="dateTime" use="required"/>
    </complexType>

    <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>
        </complexContent>
    </complexType>
    <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>
    <element name="StatusCode" type="samlp:StatusCodeType"/>
    <complexType name="StatusCodeType">
        <sequence>
            <element ref="samlp:SubStatusCode" minOccurs="0"/>
        </sequence>
        <attribute name="Value" type="samlp:StatusCodeEnumType" use="required"/>
    </complexType>
    <simpleType name="StatusCodeEnumType">
        <restriction base="QName">
            <enumeration value="samlp:Success"/>
            <enumeration value="samlp:VersionMismatch"/>
            <enumeration value="samlp:Receiver"/>
            <enumeration value="samlp:Sender"/>
        </restriction>
    </simpleType>
    <element name="SubStatusCode" type="samlp:SubStatusCodeType"/>
    <complexType name="SubStatusCodeType">
        <sequence>
            <element ref="samlp:SubStatusCode" minOccurs="0"/>
        </sequence>
        <attribute name="Value" type="QName" use="required"/>
    </complexType>
    <element name="StatusMessage" type="string"/>
    <element name="StatusDetail" type="samlp:StatusDetailType"/>
    <complexType name="StatusDetailType">
        <sequence>
            <any namespace="##any"
                processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
</schema>
```

9. References

[Needham78] R. Needham et al., *Using Encryption for Authentication in Large Networks of Computers*, Communications of the ACM, Vol. 21 (12), pp. 993-999, December 1978.

[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 ServiceRequestor (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.

[SAMLP-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.

[XMLSig] D. Eastlake et al., *XML-Signature Syntax and Processing*, http://www.w3.org/TR/xmldsig-core/, World Wide Web Consortium.

[XMLSig-XSD] XML Signature Schema available from http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/xmldsig-core-schema.xsd.

[XTAML] P. Hallam-Baker, *XML Trust Axiom Markup Language 1.0*, http://www.xmltrustcenter.org/, VeriSign Inc. September 2001.

[W3C-CHAR] http://www.w3.org/TR/WD-charreq

[UNICODE-C] http://www.unicode.org/unicode/reports/tr15/tr15-21.html

[W3C-CharMod] http://www.w3.org/TR/charmod/

[XML] http://www.w3.org/TR/REC-xml

Appendix A. Notices