Security Services Markup Language

Prateek Mishra	*Netegrity*
Phillip Hallam-Baker	*VeriSign*
Zahid Ahmed	*CommerceOne*
Alex Ceponkus	*BowStreet*
Marc Chanliau	*Netegrity*
Jeremy Epstein	*webMethods*
David Jablon	*Netegrity*
Eve Maler	*SUN Microsystems*
David Orchard	*Jamcracker*

Reviewers:

SUN Microsystems
ATG, Inc
TIBCO
Oracle
PWC

Security Services Markup Language

Version 0.7a

1 Executive Summary

Security Services Markup Language (S2ML) is a set of XML schemas and interfaces for *security services*. S2ML provides a standard description of *authentication* and *authorization* as XML request and response pairs. There are a wide range of authentication technologies in use, such as, login-password, SSL, Digital Signing, Kerberos, Smart Cards etc. There are also many frameworks for authorization including ACLs, Capabilities, Java Authorization Model etc. A major design goal for S2ML is to provide a single syntax within which a broad class of authentication and authorization techniques can be expressed and used.

S2ML identifies two key schemas --- *Name Assertions* and *Entitlements* --- that provide a foundation for *sharing* security artifacts on the internet. Traditionally, security has been viewed in the context of a transaction that is entirely contained within a single enterprise. Increasingly, transactions, whether driven by users or document flow, may authenticate at a portal or marketplace and complete through interactions at other sites. Authentication, authorization and entitlement information required to complete or enable a transaction may originate from many sites and be interpreted at other sites.

The following XML schemas and security interfaces are described in this document:

- *NameAssertion:* the result of successful authentication is a digitally signed XML assertion describing the authentication type, user and authenticator.

- *Entitlement:* is a digitally signed XML assertion consisting of a ``portable'' package of authorization data created by an issuing authority concerning an authenticated subject.

- *Authentication:* An *AuthReques*t document contains *credentials*; the result of authentication is an *AuthResponse* document containing a *NameAssertion* and may also include *Entitlements.*

- *Authorization:* An *AzRequest* document contains an *NameAssertion,* zero or more *Entitlements* and an authorization *Question*; the *AzResponse* document contains an *Answe*r and may also include *Entitlements*.

Audit, based on logging and analysis of security-related data, is a key requirement in security systems. S2ML supports audit by including information in schemas, which may be used to establish sequencing relationships between requests, responses, name assertions and entitlements over long time periods.

2 Introduction

This document describes schemas for *Name Assertions*, *Entitlement Assertions* as well as a XML-based request-response protocol for two *security services*: authentication and authorization. The protocol consists of requests and response pairs of XML documents for each service.

2.1 Structure of this document

The remainder of this document describes S2ML, the Security Service Markup Language.

3 S2ML Use Case Scenarios

S2ML can be used in environments where transactions are driven by users or services.

3.1 Scenario #1: User-Driven Transactions

Companies need a way to securely share user information as users travel across trusted partner sites. The information to be shared through single sign-on includes authentication, authorization, and profile. In this model, each partner site has its own security infrastructure. In addition to a business relationship, we assume that partners have established a trust relationship.

A typical example of a user-driven transaction environment is a large bank and its partners such as travel agencies, a 401K management company, payment services, etc. In this case, a user logs on to the large bank and can seamlessly visit the large bank's partner sites without having to re-authenticate.

Another typical user-driven transaction environment can include an Application Service Provider (ASP) aggregator that provides its users with seamless access to all the ASPs part of its trusted relationship.

In user-driven transaction environments, identity assertions can travel with the user in various ways, typically using cookies or HTTP headers.

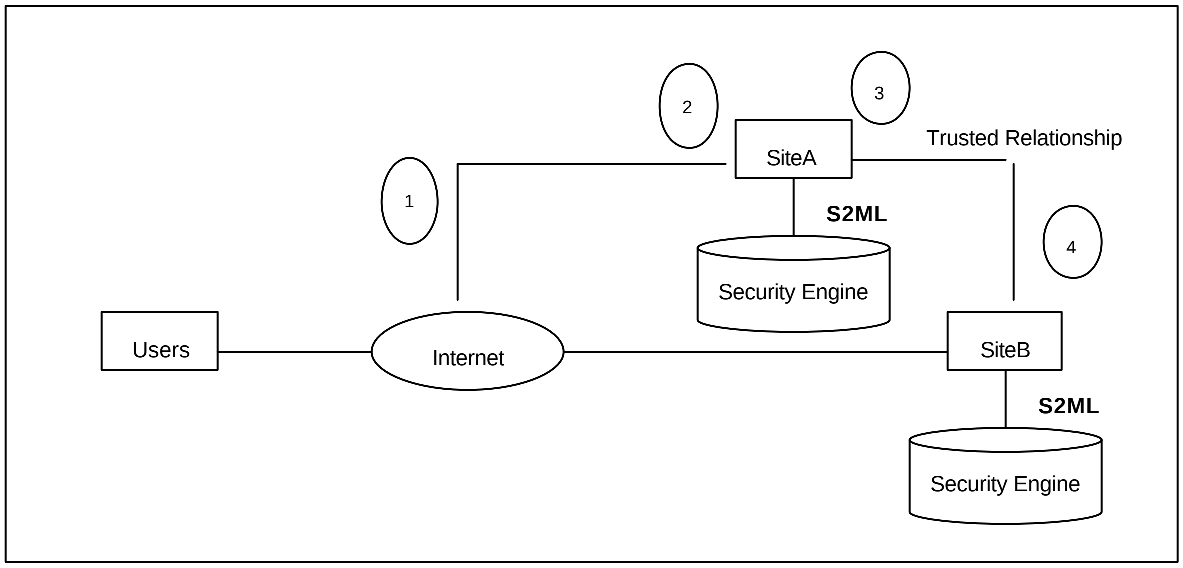
In the following example, users are able to visit SiteA and SiteB seamlessly.

(1) User logs on to SiteA and identifies herself to access SiteA's protected resources.

(2) Based on the information provided at log-in time by the user, SiteA generates an S2ML-based assertions including entitlements. S2ML assertions may be generated through the use of a S2ML conformant security engine, or, by components that transform the output of existing security engines to S2ML.

(3) User clicks on a link to a resource located at, and protected by, SiteB.

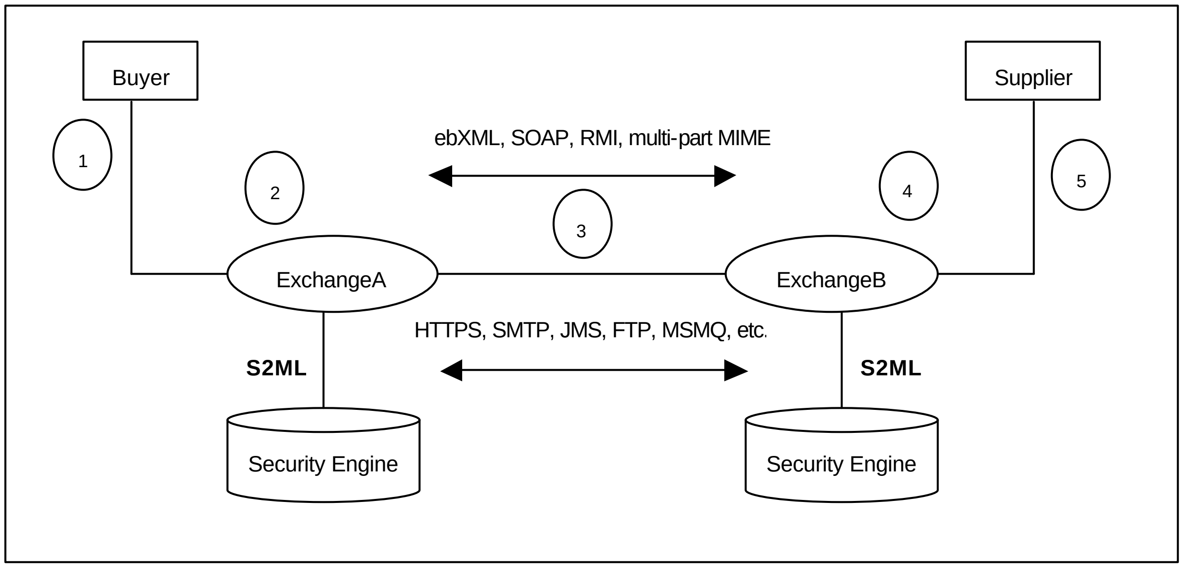
(4) User is allowed into SiteB without having to re-authenticate (information about the S2ML security token travels with the user as a HTTP header.) The information about the user authenticated at SiteA together with the entitlements from SiteA is used to complete the transaction. SiteB may also query SiteA about the user's authorizations. Thus authorization information may be both ``pushed'' from A to B, and "pulled" on an as-needed basis by B from A.



3.2 Scenario #2: Service-Driven Transactions

A typical example is multiple exchange environments as illustrated in the following figure. In this model, it is assumed that the buyer and supplier sides have a trusted relationship.

This example scenario focuses on sharing authorization entitlements between ExchangeA and ExchangeB.



(1) The buyer (an individual or a buying entity) pushes the XML document to ExchangeA. The document includes credentials, e.g., name / pwd, etc. (credentials can alternatively be discovered at the transport level.)

(2) ExchangeA authenticates the user based on credentials, and inserts a name assertion (subject description) and entitlements (for example, credit analysis information) into the document. ExchangeA can optionally remove the credentials from the document.

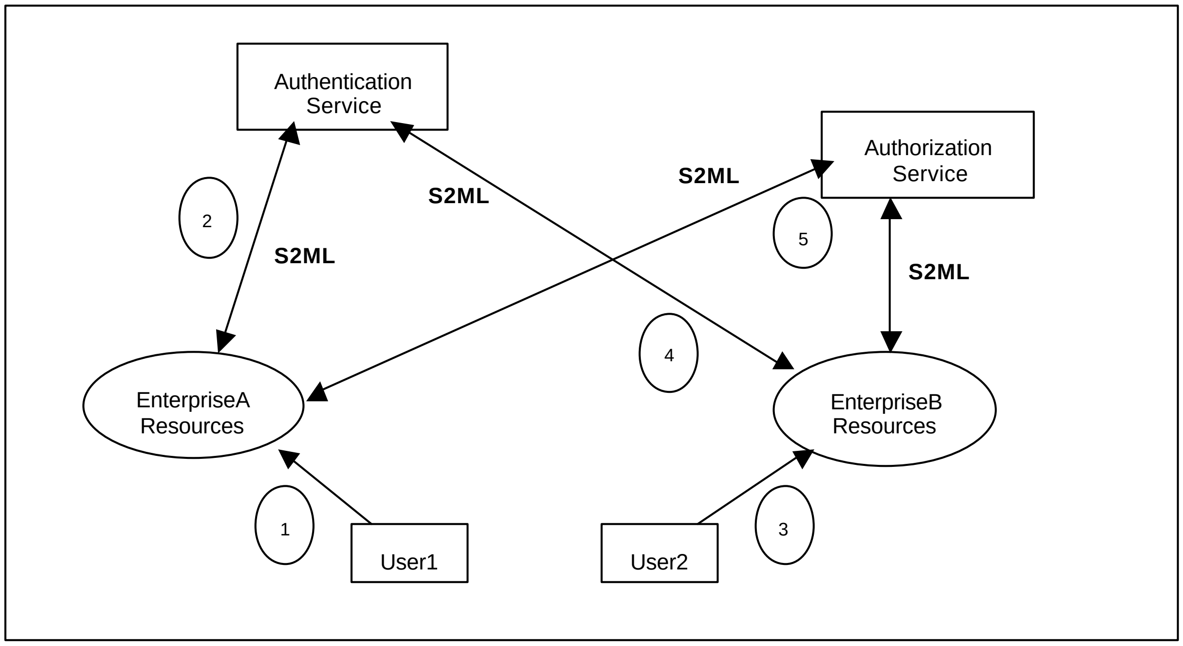
(3) The message is sent to ExchangeB using any messaging framework (SOAP, ebXML, RMI, multi-part MIME, RosettaNet, etc.) over any transport protocol (HTTPS, SMTP, JMS, FTP, MSMQ, IBMMQ, etc.)

(4) ExchangeB checks the entitlements (credit analysis information in this example) against policies stored in the security engine. If additional authorization information is needed, exchange B may ``pull'' that information from A.

(5) Based on the credit analysis information, the document is pushed to the appropriate supplier side, such as one that accepts a risk level matching the credit rating found in the provided credit analysis information.

3.3 Scenario #3: Hosted Services

In this scenario, enterprises can subscribe to remote authentication and authorization services and they can access these services through S2ML. Remote authentication and authorization services are hosted at different sites and are completely distinct. Enterprises manage their own user and policy data at the hosted service.



(1) User1 logs on to EnterpriseA.

(2) User1 is authenticated by EnterpriseA using remote authentication service.

(3) User2 logs on to EnterpriseB.

(4) User2 is authenticated by EnterpriseB using the same remote authentication service used in (2).

(5) User1 and User2 access services (i.e., business processes) at EnterpriseA and EnterpriseB. For example, a user may need specific authorization to make a purchase.

4 Architecture

We assume that one or more computational entities or *actors* are utilizing security services. Examples of actors include application servers, application programs, security services, transport and message-level interceptors etc. Various *subjects*, such as end-users, programs, actors and documents interact with the actors so as to carry out some computational process. The actors utilize security services through S2ML interfaces to ensure that the desired computational processes are secured.

Name assertions and entitlement assertions allow actors to share authentication, authorization and entitlement information. Actors insert name assertions and entitlements into transaction flows utilizing one or more *bindings*. Actors complete computational processes based on *scrutinizing* assertions and determing their validity; either by directly checking assertion validity or indirectly by calling out to an authorization engine.

S2ML places no restriction on the location, cardinality and structure of actors (and security services); the only restriction placed is that each actor MUST have a unique name (URI). All URIs used within this specification refer to absolute URIs.

Interaction between actors, and between actors and security services, involve some form of transport such as TCP, HTTP, SMTP, etc. Further, such an interaction may also involve a messaging framework such as SOAP or RMI. It is a goal for S2ML is to be transport and messaging framework neutral and to be useable with a wide variety of transports and messaging frameworks.

The interaction between actors, and between actors and security services, takes place in the context of a *trust-relationship*. In addition, depending upon the environment, there may also be a privacy requirement requiring the use of data *encryption*.

The S2ML specification distinguishes between the *minimum* security required for assertions versus those for security services. Name assertions and entitlements are "portable" pieces of information, which may travel across the internet and be scrutinized and checked for validity far from their point of origin. Therefore, they MUST be signed using the framework described in the [XML DSIG] specification. It is important to note that [XML DSIG] supports both using secret-key (e.g., HMAC) or public-key signing. When the XML encryption specifications are available, additional infra-structure will be developed within S2ML to support element-level privacy of assertions. In the interim, other standard technologies for privacy may be used.

In contrast to assertions, security services are defined by a *point-to-point* request-response protocol whose functioning is much more localized. Therefore, there is no mandatory recommendation for use of [XML DSIG]. It is recommended that standard technologies for trust and encryption be used, such as those based on:

(1) secret key encryption and signing [RC4, HMAC],

(2) transport-based security (SSL),

(3) XML digital signature, secret key or public key, XML encryption models

(4) S/MIME 2 and 3.

4.1 Name Assertions and Entitlements

Both types of assertion carry the following information:

- The set of audiences to which the assertion is addressed
- Issuer identification.
- A unique identifier.
- Time of issuance and duration of assertion validity.
- Data related to authentication (Name Assertion) or authorization (Entitlement).
- XML Digital signature which cryptographically binds issuer identity to attributes of the assertion.

A Name Assertion describes a successful authentication step:

```
<NameAssertion>
    <This>urn:authEngine32:xsde12</This>
    <Issuer>http://www.somecompany.com/authEngine32</Issuer>
    <Date>2000-10—16T12:34:120-05:00</Date>
    <Audiences>urn:all_somecompany_servers</Audiences>
    <AuthData>
         <AuthType>Login</AuthType>
         <IdentityToken>x12+21defqa$3#</IdentityToken>
    </AuthData>
    <DSIG:signature>. . . </DSIG:signature>
</NameAssertion>
```

The name assertion above indicates that actor

`http://www.somecompany.com/authEngine32`

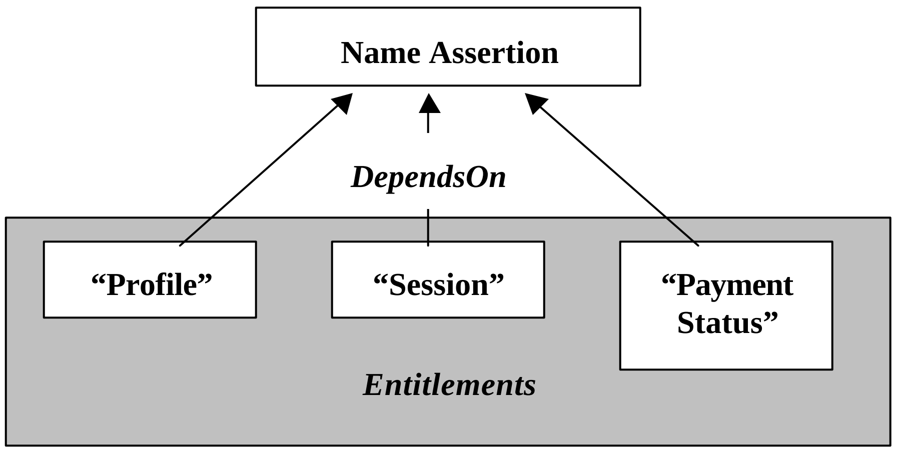
authenticated a subject, at 12:34:120 EST on the 16th of October, 2000. The assertion is scoped via the `<Audiences>` construct as directed to a certain class of actors. Elements within `<AuthData>` provide details about the authentication act: in this case, the subject provided a password and user-name, and the issuer has provided an identity token.

An entitlement assertion represents a statement made by an actor concerning an authenticated subject. For example, a server within the finance department in an enterprise may indicate a partner's payment status using the following XML fragment:

```
<Entitlement>
    <This>urn:financeDepartment:129de12</This>
    <Issuer>http://www.somecompany.com/finance/AzEngine</Issuer>
    <Date>2000-10–16T12:34:120-05:00</Date>
    <Audiences>urn:all_somecompany_partners urn:all_local_servers</Audiences>
    <ValidityInterval>
          <NotBefore>2000-10–16T19:34:120-05:00</NotBefore>
          <NotAfter>2000-10–16T20:34:120-05:00</NotAfter>
    </ValidityInterval>
    <DependsOn>urn:authEngine32:xsde12</DependsOn>
    <AzData>
        <SC:PaymentRecord xmlns:SC="http://ns.finance-vocab.org/finance">
                  <SC:TotalDue>19280.76</SC:TotalDue>
                  <SC:Over60Days>1200.00</SC:Over60Days>
                  <SC:Over90Days>10000.00</SC:Over90Days>
        </SC:PaymentRecord>
    </AzData>
    <DSIG:signature>. . . </DSIG:signature>
</Entitlement>
```

In the course of completing some transaction, such an entitlement will be *scrutinized* by one or more actors (business applications) and the transactions eventual outcome may be contingent on the validity of the scrutinized entitlements.

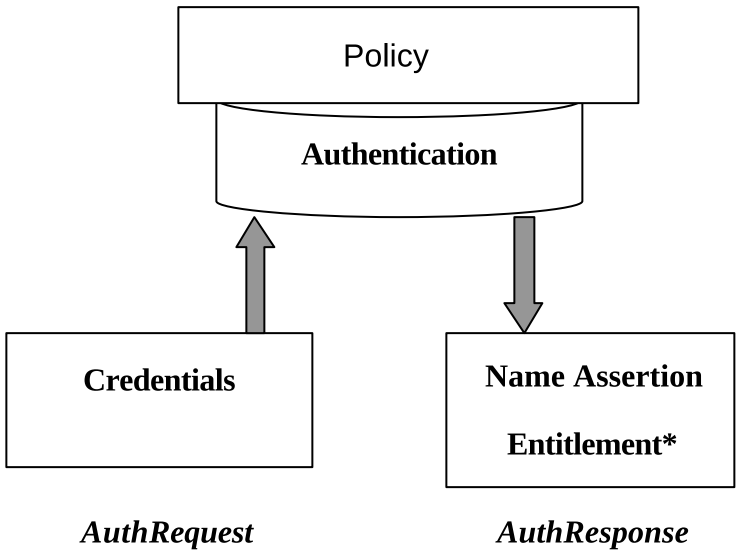
The vocabulary (elements and attributes) used to communicate entitlement data within an `<AzData>` element lie outside the scope of this specification. An entitlement must *cite* or *depend* on a name assertion. An entitlement is always a *composite assertion* and should be read as a conjunction of name assertion and entitlement.



Relationship between Name Assertions and Entitlements

4.2 Authentication (Auth) and Authorization (Az) Services

Typically, authentication services and authorization services are implemented and managed separately and this is the model developed in S2ML. From a practical point of view, there may be requirements wherein authentication and authorization need to be combined in a single step. This may be seen as a composition of the S2ML authentication and authorization steps.



In S2ML, authentication is defined as a service which consumes subject credentials and, if successful, returns a name assertion and zero or more entitlements appropriate to the subject. The name assertion is a description of the subject based on valid credentials at a certain point in time. Any entitlements returned from the authentication service, provide additional information about the subject, such as profile information or a session description.

Consider the following authentication request: an actor has created an `<AuthRequest>` message containing login credentials obtained from a subject. The request includes a unique identifier. The credentials may have been obtained by the actor in a variety of different ways: direct interaction with a user, extracted from a document etc.

```
<AuthRequest>
    <This>urn:JavaServletPlugInRequest:988</This>
    <Date>2000-11–16T11:34:120-05:00</Date>
    <Credentials>
         <Login>
          <Name>SomeUser</Name>
          <Password>aSecret</Password>
          </Login>
     </Credentials>
</AuthRequest>
```

S2ML 1.0 describes schemas for four types of credentials (Section 5.3.5): no credentials, login, X509 certificates and Public Keys. The `<Credentials>` element also permits the use of foreign namespaces through the use of the `<Any>` element. This may be used as the means for extension to other authentication schemes.

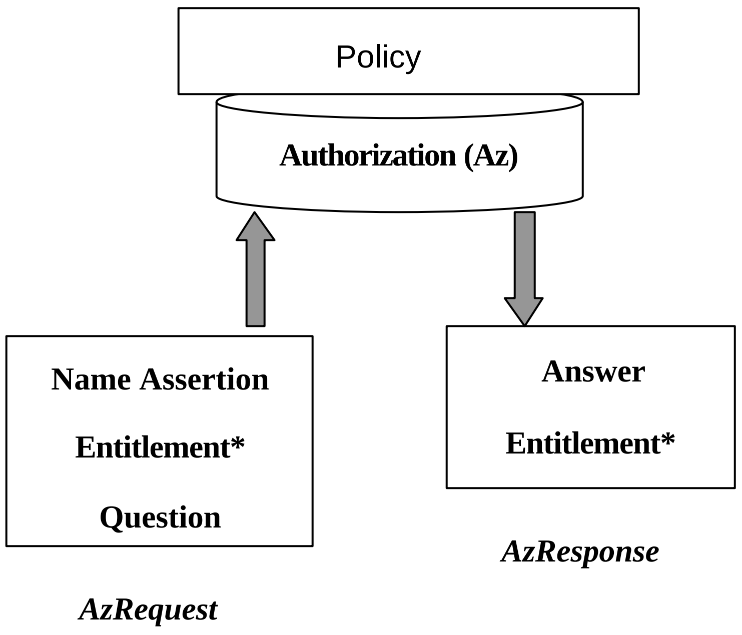
The authentication engine responds with an `<AuthResponse>` message; if authentication succeeds, the message includes a name assertion describing the authentication type and subject attributes.

```
<AuthResponse>
    <This>urn:MainAuthServer:0981</This>
    <Date>2000-11–16T12:34:120-05:00</Date>
    <Request>urn:JavaServletPlugInRequest:988</Request>
    <Result>Success</Result>
    <NameAssertion>
        <This>urn:authEngine32:xsde12</This>
        <Issuer>http://www.somecompany.com/authEngine32</Issuer>
        <Date>2000-11–16T12:36:120-05:00</Date>
        <Audiences>urn:all_somecompany_servers</Audiences>
        <ValidityInterval>
          <NotBefore>2000-11–16T19:34:120-05:00</NotBefore>
          <NotAfter>2000-11–16T20:34:120-05:00</NotAfter>
        </ValidityInterval>
        <AuthData>
         <AuthType>Login</AuthType>
         <UserHandle>
              <Directory>XJN-Q3</Directory>
              <X509.DN>uid=bjensen,ou=people,dc=airius,dc=com</X509.DN>
         </UserHandle>
        </AuthData>
        <DSIG:signature>. . . </DSIG:signature>
    </NameAssertion>
</AuthResponse>
```

S2ML 1.0 provides schemas for four types of subject attributes (Section 5.3.6) which may be contained within an `<AuthData>` element:

- `<UserHandle>` element, consisting of a string user-store name and an X.509 Distingushed name string,
- `<IdentityToken>` element, consisting of a string,
- X509 Certificate,
- Public Keys.

The `<AuthData>` element also permits the use of foreign namespaces through the use of the `<Any>` element. This may be used as the means for extension to other forms of subject description.



Authorization is a central concept in S2ML. Providing a description for authorization requires distinguishing between the basic information flow in authorization versus the existing variety of specific authorization models, including those based on ACLs, Capabilities, Java Authorization model, Rules-based models, etc. For all of these cases, however, it is possible to develop a model based on information flow:

- An authorization *question* is posed, in the context of an authenticated subject. This can take many forms as in:

  Can user X access resource R?
  OR
  Can user X withdraw $10,000 from account A?

Sometimes, there may be additional information available about user X, such as the user's profile. In such a case, the authorization question is scoped by the user identity AND the entitlements specifying the user profile.

- The authorization engine responds with an *Answer:*

  Yes, user X may access resource R.
  OR
  Yes, user X may withdraw $10,000 from account A.

  Such an answer may just have local scope, in that it is used immediately at the point of enforcement and then discarded. More broadly, however, there may also be components to the answer which are meaningful to other applications, such as the entitlements:

  The locator number for user X for accessing R is 17865X.
  User X is a platinum-class account holder with over $100,000 in funds.

Our approach to the diversity of authorization models is to use a `<AzModel>` attribute for the `<Question>` and `<Answer>` element which binds the contents of these elements to a specific authorization model. The `<AzModel>` attribute takes a URI value.

S2ML describes only one particular authorization model with URI:

http://az.s2ml.org/SimpleAz

This model describes a class of authorization questions of the form:

```
        VERB        Resource
```

model (e.g., GET http://www.somecompany.com/index.html) and answers of the form `success` or `failure`.

Authorization services MAY implement one or more authorization models; each will have its own vocabulary and associated `AzModel` URI. An `AzModel` error MUST be returned by an authorization service, if a question drawn from an unknown `AzModel` is presented in an `AzRequest` element. An authorization service SHOULD implement the SimpleAz model in addition to any other implemented models.

An AzRequest MUST include an name assertion and MAY include one or more entitlements. An AzRequest MUST include a <Question> element.

```
<AzRequest>
    <This>urn:Interceptor1AzRequest:988</This>
    <Date>2000-10—16T12:34:120-05:00</Date>
    <NameAssertion>. . . </NameAssertion>
    <Question AzModel="http://az.s2ml.org/SimpleAz">
           <ResourceContext>
                  <Method>urn:GET</Method>
                  <Resource>http://www.myserver.com/index.html</Resource>
            </ResourceContext>
    </Question>
</AzRequest>
```

An `<AzResponse>` MUST contain an answer element and MAY contain one or more entitlements. The `<Answer>` element contains a response to the authorization question posed in `<AzRequest>.` One or more entitlements may be returned from an authorization request; for example, when a user is authorized to access a commerce application, the user's locator number and payment status may be returned within an entitlement.

```
<AzResponse>
    <This>urn:GeneralPurposeAzEngine:908a</This>
    <Request>urn:Interceptor1AzRequest:988</Request>
    <Date>2000-10—16T12:34:120-05:00</Date>
    <Entitlement>. . . </Entitlement>
    <Answer AzModel="http://az.s2ml.org/SimpleAz">
           <Result>Success</Result>
     </Answer>
</AzResponse>
```

4.3 Assertion Validity

Scrutinizing actors will need to determine the validity of both name assertions and entitlements. Validity is defined in the context of business relationship with the issuer and security policies in place at the actor scrutinizing the assertion. Minimally, the following conditions MUST be evaluated by an actor scrutinizing as assertion:

1. The issuer is trusted by the actor,

2. Issuer digital signature is valid at time of scrutiny and binds to required elements in the assertion,

3. The time period for which the assertion is being scrutinized must lie within the time period specified by the `<ValidityInterval>` element.

4. The business relationship between the actor and issuer references at least one of the `<Audience>` elements.

A compound assertion (entitlement) is valid iff it meets the above rules AND the cited name assertion is valid.

4.3.1 Audience Restriction

Assertions MAY be addressed to a specific audience. Although a party that is outside the audience specified is capable of drawing conclusions from an assertion, the issuer explicitly makes no representation as to accuracy or trustworthiness to such a party.

- Require users of an assertion to agree to specific terms (rule book, liability caps, relying party agreement)

- Prevent clients inadvertently relying on data that does not provide a sufficient warranty for a particular purpose

- Enable sale of per-transaction insurance services.

An audience is identified by a URI that identifies to a document that describes the terms and conditions of audience membership.

Each actor is configured with a set of URIs that identify the audiences that the actor is a member of, for example:

`http://cp.verisign.test/cps-2000`
Client accepts the VeriSign Certification Practices Statement

`http://rule.bizexchange.test/bizexchange_ruebook`
Client accepts the provisions of the *bizexchange* rule book.

An assertion MAY specify a set of audiences to which the assertion is addressed. If the set of audiences is the empty set there is no restriction and all audiences are addressed.

4.4 Scope and Limitations

It is not a goal for S2ML to propose any new cryptographic technologies or models for security; instead, emphasis is placed on description and use of well known security technologies utilizing a standard syntax (markup language) in the context of the internet.

This document does not describe services or markup for security services such as non-repudiation. These are considered to be outside the scope of S2ML 1.0.

Authentication methods in S2ML 1.0 are limited to login, based on name and password, validation of X509v3 certificates and public keys.

Challenge-response authentication protocols are outside the scope of S2ML 1.0.

Protocols for creation and management of user sessions are outside the scope of S2ML 1.0.

5 Message Set

5.1 URI Naming Infrastructure

The S2ML Architecture makes extensive use of URIs to identify assertions, actors and audiences. The use of a URI as an object *identifier* is a superset from the use of a URI as an object *locator*. S2ML introduces objects such as audiences and authorization roles that carry distinct semantics even though there is no means of locating or even resolving them. Appendix B describes rules for URI equality.

5.2 Common Syntax

The following data elements are used in the message set:

5.2.1 `This`

The `This` element specifies a unique label for the assertion by means of a URI. It is defined by the following XML schema:

```
<element name="This" type="uriReference"/>
```

5.2.2 `Issuer`

The `Issuer` element specifies the issuer of the assertion by means of a URI. It is defined by the following XML schema:

```
<element name="Issuer" type="uriReference"/>
```

5.2.3 `ValidityInterval`

The `ValidityInterval` structure specifies limits on the validity of the assertion.

```
<complexType name="ValidityInterval">
   <all>
      <element name="NotBefore" type="timeInstant" minOccurs="0" />
      <element name="NotAfter" type="timeInstant" minOccurs="0" />
   </all>
</complexType>
```

Member	Type	Description
`NotBefore`	`timeInstant`	Time instant at which the validity interval begins
`NotAfter`	`timeInstant`	Time instant after which the validity interval has ended

The `NotBefore` and `NotAfter` elements are optional. If the value is either omitted or equal to the start of the epoch it is unspecified. If the `NotBefore` element is unspecified the assertion is valid from the start of the epoch until the `NotAfter` element. If the `NotAfter` element is unspecified the assertion is valid from the `NotBefore` element with no expiry. If neither element is specified the assertion is valid at any time.

All time instances SHOULD be interpreted in Universal Coordinated Time unless the parties concerned have agreed in advance to use a different time standard. Implementations MUST NOT generate time instances that specify leap seconds.

For purposes of comparison the time interval `NotBefore` to `NotAfter` begins at the earliest time instant compatible with the specification of `NotBefore` and *has ended* after the earliest time instant compatible with the specification of `NotAfter`.

For example if the time interval specified is *`day`*`T12:03:02` to *`day`*`T12:05:12` the times `12:03:02.00` and `12:05:12.9999` are within the time interval. The time `12:05:12.0001` is outside the time interval.

5.2.4 DateTime Date

The DateTime instant MUST fully specify the date.

```
<element name="Date" type="timeInstant"/>
```

5.2.5 Audiences

The `Audiences` element specifies a set of audiences to which the assertion is addressed. The element is defined by the following XML schema:

```
<simpleType name=”listOfUriRefs”>
   <list itemType=”uriReference”>
</simpleType>

<element name="Audiences" type="listOfUriRefs">
```

5.2.6 DependsOn uriReference

The `DependsOn` element allows an assertion to refer to or cite another assertion, thereby forming a compound assertion. A compound assertion is valid if only if each component assertion is valid.

```
<element name="DependsOn" type=“uriReference"/>
```

5.2.7 Request

The request element is used as part of the response structure to track the URI of the request object.

```
<element name="Request" type=“uriReference"/>
```

5.2.8 ResultCode

The enumerated type `ResultCode` is used to return result codes from each interface. It has the following possible values:

`Success`
The operation succeeded.

`Failure`
The operation failed for unspecified reasons.

`ResultCode` is defined as:

```
<simpleType name="ResultCode" base="string">
   <enumeration value="Success"/>
   <enumeration value="Failure"/>
</simpleType>
```

5.3 Authentication

5.3.1 NameAssertion

```
<element name="NameAssertion">
<complexType>
   <all>
      <element name ref = “This"/>
      <element name ref = “Issuer"/>
      <element name ref = “Date"/>
      <element name ref = “Audiences” minOccurs=”0” />
      <element name ref = “ValidityInterval" minOccurs=”0”/>
      <element name ref = "AuthData"/>
      <element name ref = “DSIG:signature”>
   </all>
</complexType>
</element>
```

Where the schema elements have the following use:

Identifier	Type	Description
`This`	`URI`	Assertion identifier. MUST be present. MUST satisfy the uniqueness property
`Issuer`	`URI`	
`Date`	`timeInstant`	Time instant of issue
`AuthData`		Information generated by authentication step.
`ValidityInterval`	`ValidityInterval`	Optional
`Audiences`		Optional
`DSIG:signature`		Enveloped digital signature binding issuer identity to required assertion attributes

5.3.2 Request Message

The following schema defines the request message:

```
<element name="AuthRequest">
   <complexType>
      <all>
         <element ref = “This”/>
         <element ref = “Time”/>
         <element ref = “Credentials”/>
      </all>
   </complexType>
</element>
```

5.3.3 Response Message

The following schema defines the response message:

```
<element name="AuthResponse">
   <complexType>
      <sequence>
         <element ref = “This”/>
         <element ref = ”Time”/>
         <element ref = “Request”/>
         <element name = “Result" type=”ResultCode”/>
         <element ref = “NameAssertion” minOccurs=”0”/>
         <element ref = “Entitlement” minOccurs=”0” maxOccurs=”unbounded”/>
      </sequence>
   </complexType>
```

```
</element>
```

The `<request>` element contains the unique identifier of the `<AuthRequest>` element for which this `<AuthResponse>` element has been created.

5.3.4 Login

The Login element must contain a name and password pair; it may also contain an optional realm or domain element.

```
<element name=”Login”>
         <all>
           <element name=”Name” type=”string”/>
           <element name=”Password” type=”string”/>
           <element name=”Domain” type=”string” minOccurs=”0”/>
         </all>
</complexType>
</element>
```

5.3.5 Credentials

The `Credentials` element may contain any one of four standard elements, or an element derived from a namespace other than S2ML. The `Nocreds` element indicates that no credentials are being provided.

```
<element name="Credentials">
<complexType>
       <choice>
           <element ref =”Login”/>
           <element ref=”DSIG:X509Data”/>
           <element ref=”DSIG:KeyValue”/>
           <any namespace=”##other”/>
           <element Nocreds/>
       </choice>
</complexType>
</element>
```

5.3.6 AuthData

The `AuthData` element encodes the result of a successful authentication step. The `AuthType` element describes the type of credentials that presented for authentication. Credentials are mapped into one of four standard forms: `UserHandle`, `IdentityToken`, `DSIG:X509Data, DSIG:KeyValue`.

```
<element name="AuthData">
<complexType>
     <sequence>
       <element ref = “AuthType”>
       <choice>
           <element ref = “UserHandle”/>
           <element ref = “IdentityToken”>
           <element ref= “DSIG:X509Data”/>
           <element ref=”DSIG:KeyValue”/>
           <any namespace= “##other”/>
       </choice>
     <sequence>
</complexType>
</element>
```

5.3.7 AuthType

```
<element name="AuthType">
<complexType>
       <choice>
           <simpletype base=”string”>
               <enumeration value="Login"/>
               <enumeration value="Nocreds"/>
               <enumeration value="X509Data"/>
               <enumeration value=”KeyValue” />
           </simpleType>
           <any namespace= “##other”/>
       </choice>
</complexType>
</element>
```

5.3.8 UserHandle

Element `UserHandle` represents the case wherein credentials are mapped to an entry within a directory or user store. Element `X509.DN` MUST take the form of an X.509 Distinguished Name [X.509], for example:

uid=bjensen,ou=people,dc=airius,dc=com

```
<element name=”UserHandle”>
<complexType>
<all>
     <element name = “Directory” type = “string” />
     <element name = “X509.DN” type = “string” />

</all>
</complexType>
</element>
```

5.3.9 IdentityToken

```
<element name=”IdentityToken” type=”string”/>
```

5.4 Authorization

5.4.1 Entitlement

The Entitlement (Assertion) element

```
<element name=”Entitlement”>
<complexType>
   <all>
     <element name ref =”This”/>
     <element name ref =”Issuer”/>
     <element name ref = “Date”/>
     <element name ref = “Audiences” minOccurs=”0”/>
     <element name ref = “DependsOn”>
     <element name ref =”AzData” />
     <element name=”ValidityInterval” type=”ValidityInterval” minOccurs=”0”/>
     <element name ref = “DSIG:signature” />
   </all>
</complexType>
```

Where the schema elements have the following use:

Identifier	Type	Description
`This`	`String`	Assertion identifier. MUST be present. MUST satisfy the uniqueness property
`Date`	`timeInstant`	Time instant of issue
`Issuer`	`uriRef`	
`DependsOn`	`uriRef`	Link to Name Assertion
`ValidityInterval`	`ValidityInterval`	Optional
`Audiences`		Optional
`DSIG:signature`		Enveloped digital signature binding issuer identity to assertion attributes

5.4.2 Request Message

The following schema defines the request message:

```
<element name="AzRequest">
   <complexType>
      <sequence>
      <element ref = “This”/>
      <element ref = “Time”/>
       <element ref = “NameAssertion”/>
      <element ref = “Question”/>
      <element ref = “Entitlement” minOccurs=”0” maxOccurs=”unbounded”/>
     </sequence>
   </complexType>
</element>
```

5.4.3 Response Message

The following schema defines the response message:

```
<element name="AzResponse">
  <complexType>
     <sequence>
      <element ref = “This”/>
      <element ref = “Time”/>
      <element name=”Request” type=”uriRef”/>
      <element name ref = “Answer”/>
      <element name ref = “entitlement” minOccurs=”0” maxOccurs=”unbounded”/>
     </sequence>
   </complexType>
</element>
```

5.4.4 ResourceContext

```
<element name="ResourceContext">
<complexType>
   <all>
      <element name="Resource" type="uriReference"/>
      <element name="Method" type="uriReference"/>
   </all>
</complexType>
</element>
```

Where the sub-elements have the following meaning

Identifier	Type	Description
`Resource`	`URI`	Resource Name
`Method`	`URI`	Verb

5.4.5 AzData

```
<element name="AzData">
<complexType>
   <all>
     <any namespace=”##other”/>
   </all>
</complexType>
</element>
```

5.4.6 Question

```
<element name="Question">
<complexType>
   <choice>
     <element ref = “ResourceContext”>
     <any namespace=”##other”/>
   </choice>
   <attribute name=”AzModel” type=”uriRef” />
</complexType>
</element>
```

5.4.7 Answer

```
<element name="Answer">
<complexType>
   <all>
     <element name="Result" type="ResultCode"/>
     <any namespace=”##other”/>
   <all>
   <attribute name=”AzModel” type=”uriRef” />
</complexType>
</element>
```

6 Binding

6.1 HTTP Binding

In many user-driven scenarios there is a need to communicate security information through HTTP headers as discussed in [User-driven Use-Case]. In such a case, assertions originating from one site may need to be communicated to another site through HTTP headers. As S2ML assertions may be of variable size and HTTP headers are strongly size constrained, this specification describes a system in which *unambiguous references* to S2ML assertions are conveyed through HTTP headers. Using such references sites may retrieve S2ML assertions from other sites through means that lie outside the scope of this specification.

The S2MLheader HTTP header follows the standard HTTP header format as in [RFC2068].

```
S2MLheader “ : “ <encrypted-payload>
```

The encrypted payload is comprised of a reference to a *single* assertion. The payload is constructed in the following manner:

20 octets: *Sender Description Digest* A 20-byte SHA1 hash of the Sender URI
20 octets: *Message Digest* A 20-byte SHA1 hash of contents of `<This>` element of an assertion.

The payload is encrypted using 128-bit secret key encryption based on the US AES standard.

The encrypting and decrypting cipher is the AES(*) cipher in CBC mode with 128-bit blocks. Encryption and decryption will use a constant initialization vector of 16 zero bytes. The input to the cipher is a set of three 16-byte blocks formatted from the following concatenated blocks:

```
4 bytes   random header
20 bytes  message digest value
20 bytes  sender description digest value
4 bytes   reserved, must be set to 0, 0, 0, 1
```

The three 16-byte blocks that are output from the AES-CBC-128 cipher form a 48-byte message.

The sending site will store and manage a table of assertions for which references have been exported outside the site. The table is indexed by the Message Digest element of each assertion. The receiving site will decrypt and verify the S2ML header payloads. Based upon the sender description digest, it will contact the sending site and "pull" the relevant assertions.

(*) At the time of this proposal, the selection of Rijndael as the AES cipher is not yet finalized by NIST.

Security of cipher construction

Using digest values for the Sender URI and Message fields serves two distinct purposes. The primary purpose is to create a unique handle for these fields, where it is not feasible for anyone to construct another field value that hashes to the same digest value. The SHA1 hash function is suitable to achieve this cryptographic property. The secondary purpose of the digest value is as a database lookup key. SHA1 is certainly more than sufficient for this purpose.

A standard cryptographic envelope using a 128-bit key would expand the message by at least 32 bytes, including a 16-byte initialization vector (IV) and a 16-byte message authentication code. In contrast, our cipher construction is optimized to save a few bytes, at the expense of some (perhaps torturous) analysis. The cipher text is only 8 bytes larger than the clear text, where 4 bytes act as a crude message authenticator, and another 4 bytes act as a simple message obfuscator. While this construction does give us full 128-bit privacy protection, it does not strongly authenticate the message, nor does it strongly guarantee that two identical plain texts won't appear as equal. The rest of this section explains why we can "get away" with this.

We have no compelling need for strong message authentication, because it is impossible to create a valid forged message through cipher-text manipulation. This is guaranteed because our plaintext consists only of cryptographic digests.

Similarly, while we enjoy the benefits of 128-bit message privacy, we have no particularly strong need to prevent enemies from detecting that two enciphered messages correspond to identical plain text. Sure we'd like to hide this fact, but if one can tell with a one in 2^32 chance that two plaintexts are equal, this is not a major threat. Simple traffic analysis will almost certainly pose a much greater threat.

So, of the 8 extra bytes, 4 are set to zero, and used to provide a weak form of message authentication, which is largely used to detect accidental corruption. Another 4 bytes are used to provide a random message ID, which provides for a reasonable level of information hiding in this application.

Note that of the two digest values, we place the message digest first, since it will be varying much more than the Sender URI digest, and thus offer added protection against an enemy occasionally noticing that two messages are the same. If the Sender URI was positioned first, one might see with a 1/2^32 chance, that two different messages came from the same sender, regardless of the whether the message itself was distinct.

Also note that the value of the last byte is "1" to be compatible with standard message padding schemes (=== name them), which require at least one padding byte to be present.

6.2 MIME Binding

MIME and particularly Multipart-MIME are very commonly used in XML messaging systems. Hence, a S2ML document instance can be packaged into various MIME based enveloping schemes, including S/MIME which supports multipart/signed content types.

The purpose of the following discussion is to specify how S2ML documents instances can be applied into a MIME messaging protocol. However, this discussion is independent of the specific MIME messaging protocol used. Our viewpoint is that the S2ML MIME binding should provide a reasonable “default” binding for messaging based on MIME packaging. Individual messaging frameworks may provide more specific ways to include S2ML fragments and assertions.

Hence, in the process of this discussion we note what is required and what is optional from the standpoint of consistently publishing, processing, and incorporating S2ML documents into a MIME message processing system, e.g., used in a B2B messaging protocol.

Requirements for Multipart MIME Packaging

- **Use of multipart/mixed**

  S2ML documents such a Credential, Identity Assertions, and Entitlement can be packaged using MIME *multipart/related* into a single MIME package. This security document can be included into a *multipart/mixed* envelope as a MIME part or as an attachment;

- **Encryption**

  The S2ML document may be optionally encrypted if S/MIME enveloping (is employed) using *application/pkcs7-mime* content-type. E.g., the Credential document that includes login password may need to be protected over insecure transports. Entitlement information may also need to be protected if it is being processed over multiple sites over varying transport protocols.

- **Signed S2ML Documents**

  Since XML DSig will be employed to sign and endorse the S2ML document instance, it is not required that S2ML parts in a MIME envelope be additionally signed.

Logical Schema for Packaging S2ML in a MIME Message

6.1.1 MIME Header

Header Component	**Description**
Header attributes	MIME Message Header properties

6.1.2 S/MIME Message Top-level Signature Block

There is a need to package the message payload and the S2ML Security Documents that have been created as a result of a successful validation of the message originator and/or the message. The specific packaging structure is beyond the scope of S2ML specification since it depends on the messaging protocol. However, in all messaging protocol cases, we want to make sure that the business document(s) (i.e., message payload) is linked with the S2ML security assertions generated by the security serviice provider.. There are a number of ways this can be accomplished and the corrrect packaging approach depends on the specific messaging protocol employed.

Here we demonstrate an example of packaging the S2ML Security Documents Block with the message payload via a top-level message signature generated over the multipart/related S2ML Security Documents and message payload parts.

Signature Component	**Description**
Signature *Signature Format*	Generated using the private signature key of the some trusted party (and/or message over both the S2ML Security Documents Block and the message payload. The purpose of the outer signature is to ensure that the Name assertion and any entitlements associated with message originator are combined with the original message payload such that no intermediary party can replace the original S2ML security properties (e.g., during message transmissions) with a false set of S2ML documents. ***Note:*** This signature block may be optional for some messaging protocols since 1) their will be transmission security (e.g., SSL) and 2) their will be a signature on each S2ML document by a trusted security provider that authenticates the message which may be sufficient in some messaging apps. *multipart/signed*
Public Certificate Chain	The public certificate corresponding to the end-entity that is signing the message containing both the S2ML Security Documents and original message payload. The first certificate in the chain is the actual public identity of the originating party; the rest of the chain elements are used to establish trust . e.g., trusted third party CA.

6.1.3 S2ML Security Documents Block

The message authentication service provider will typically generate one or more S2ML documents which will represent the following types of security information:

- application authentication information, i.e., Credential of the message originator

- authenticated identity of the message originator, i.e., name assertions created/endorsed by a trusted security service provider;
- entitlements associated with the message originator, e.g., access rights or membership domain that are validated/endorsed by a security service provider;

The Credential document is always created and included into MIME message package by the message oroginating party. The Name assertion and any entitlements are included into the MIME message by a trusted security service provider that manages authentication and authorizations services. The messaging application will package these elements together into a *multipart/mixed* part of the MIME message envelope such that:

- **Name Assertions Linked with Entitlement** *(REQUIRED)*

  Name assertion document is always linked with their corresponding one or more Entitlement

  assertions documents;

- **S2ML Security Documents Block is Protected** *(OPTIONAL)*

  Signing the S2ML security block will ensure that this part is not tampered with; for example, to ensure that none of the entitlement documents are discarded. This will be highly useful if there is no transmission-level security and/or there are multiple message processing parties involved in the B2B transaction.

S2ML Component	**Description**
Credential/Name Assertion/Entitlements *Content Type*	The S2ML document instance (of content type *text/xml)* is part of a *multipart/mixed* message part. *Multipart/mixed*
Encrypted S2ML Envelope *Content Type*	The S2ML document instance will be optionally encrypted with the public key of the receiving party. The resulting packaged in an *application/pkcs7-mime* envelope layer.

Signed and encrypted S2ML documents	The S2ML Security Documents Block may also optionally be signed. ***Note:*** The entity signing the S2ML Security Document block maybe distinct from the entities that have signed the name and entitlement assertions. E.g., this signature may be the messaging application that uses the security services that validated/created the S2ML documents.
Content Type	*The resulting MIME content type application/pkcs7 signature.*

6.1.3.1 Example of S2ML Security Documents Block

```
    Multipart/MIXED;,,,,; boundary="xxxxxxxxx"

-"xxxxxxxxxxxxxx"

Name-Assertion Document

    -"xxxxxxxxxxxxx"

    Entitlement-Document #1

    -"xxxxxxxxxxxxx"

    Entitlement-Document #1

    -"xxxxxxxxxxxxx"
```

6.1.4 Encrypted/Signed Payload Block

Payload Component	**Description**
Payload component	Payload components refer to the primary message body.
Encrypted Payload Envelope	The payload will be optionally packaged in an *application/pkcs7-mime* envelope layer.
Signed and encrypted payload	The payload can also optionally be signed and the resulting envelope is of the MIME content type *application/pkcs7-signature*

6.2 ebXML Binding

TBD

6.3 SOAP Binding

Binding Notes:
- It is up to the application about whether the 'mustUnderstand' attribute will be applied to the headers.
- It is up to the application about whether the 'Actor' attribute will be applied to the headers.
- All entries in the Header MUST be namespace qualified (requirement of SOAP 1.1)
- Until XML encryption standard becomes available, no standard technique is available to precisely to encrypt the S2ML headers. However, the entire SOAP message can be placed in MIME packaging and S/MIME technology utilized for encryption.

```
* Passing around IdentityAssertion and Entitlement:

<soap-env:Envelope
  xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
  soap-env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

  <soap-env:Header>
    <s2ml:NameAssertion xmlns:s2ml="http://ns.s2ml.org/S2ML" />
    <s2ml:Entitlement xmlns:s2ml="http://ns.s2ml.org/S2ML" />
    <s2ml:Entitlement xmlns:s2ml="http://ns.s2ml.org/S2ML" />
  </soap-env:Header>

  <soap-env:Body>
    <message_payload/>
  </soap-env:Body>

</soap-env:Envelope>
```

7 Conformance

Four levels of conformance are defined:

1. A security system is a **consumer** of S2ML it can provide authorization decisions based on name assertions and entitlements generated elsewhere.

2. A security service which is a S2ML consumer may also provide an S2ML conformant authorization service. This type of service cannot create any entitlements, but can read name assertions and entitlements (**Read-only S2ML Az**) and determine their

validity.

3. A security system is a **consumer** and **producer** of S2ML if it can both produce and consume name assertions and entitlements.

4. A security system which is a S2ML consumer-producer may also provide a S2ML conformant authentication or authorization service (**S2ML Auth, S2ML Read-Write Az**).

Appendix A URI Equality: Lexical Comparison

The equality function used on URIs is strictly lexical and are applied *without reference* to the semantics of the underlying URI name space. The rules for lexical comparison of URIs described here differ in some respects to the rules for semantic equivalence of URIs specified in RFC 2396 [RFC2396].

Use of lexical comparison functions ensures that the comparison functions are defined even though the application may not understand the resolution semantics of the underlying name space. The complexity of client implementations is reduced through application of the following rules:

- The forward slash character '`/`' is *always* interpreted as a separator for different levels in the name space hierarchy. No other character is interpreted as a separator.
- Comparison is always performed within the ASCII character set encoding of the URI.
- Characters describes as escaped, reserved and unreserved in RFC 2396 are always regarded as being so.

RFC 2396 defines rules for semantic equivalence of URIs. To simplify client implementation the following forms of URI are differentiated:

- A URI that specifies the default port explicitly is NOT equivalent to a URI that specified the default port implicitly (i.e. `http://site.test/` is distinct from `http://site.test:80/`).

Differentiating between explicitly and implicitly defined port numbers ensures that lexical comparison is consistent even though a client may not understand the resolution semantics of a URL scheme.

The following forms of URI are never differentiated:

- A URI that does end in a forward slash character '`/`' is directly equivalent to the same URI with a slash character appended at the end.
- A URI in which a character is escaped is directly equivalent to one in which the character is not escaped. Where more than one means of character escape is defined for the same character no distinction is made on the basis of the escape mechanism chosen.

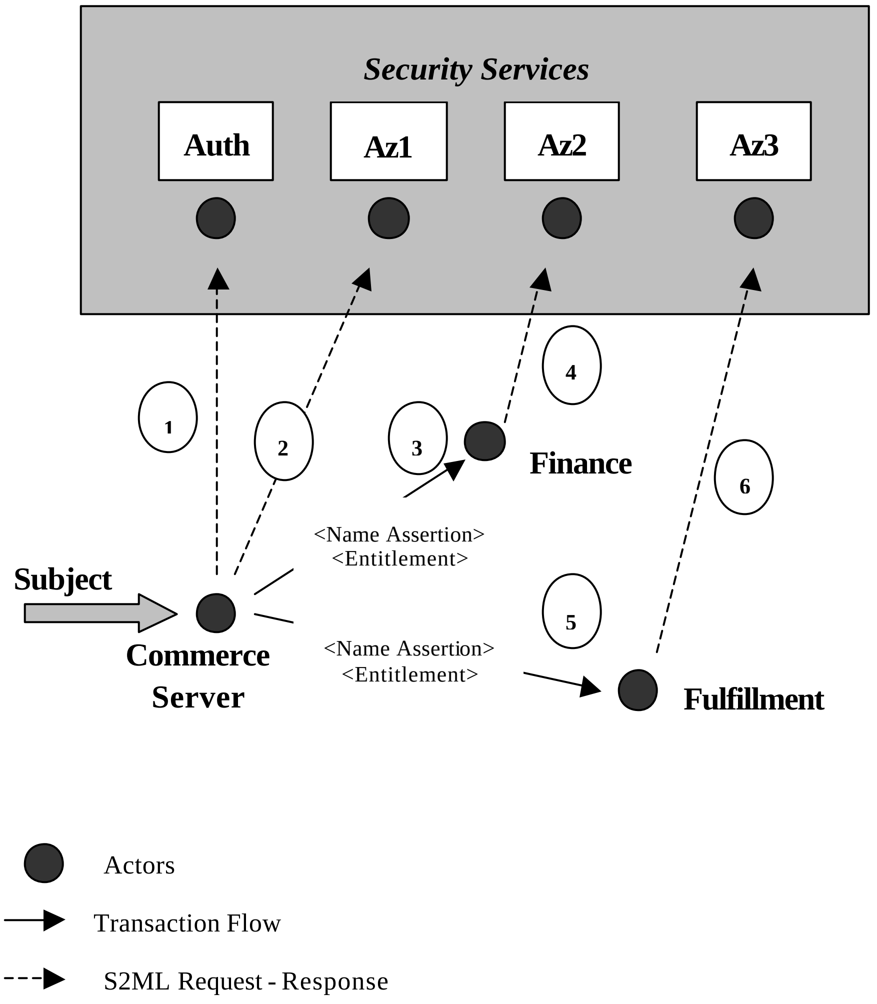
Applying these rules the following URIs are not differentiated.

```
http://site.test/my+resource
http://site.test/my%20resource
```

```
http://site.test/my+resource/
http://site.test/my%20resource
```

Appendix B: Securing a Multi-Step Transaction

In this example, we follow a multi-step transaction which may involve one or more enterprises. A subject is interacting with a commerce engine so as to complete some business transaction. The commerce engine further interacts with finance and fulfillment servers for sub-parts of the transaction. Each server may be located in a different enterprise or within a different administrative area within a single enterprise. A single remote `Auth` server is utilized for authentication; multiple authorization servers are utilized for checking subject authorization.



WORK THROUGH THE FLOW ---- TBD

Appendix C References

[RFC2396] T. Berners-Lee, R. Fielding and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax* RFC 2396, August 1998, Internet Engineering Taskforce. http://www.rfc-editor.org/rfc/rfc2396.txt.

[XML-SIG] D. Eastlake, J. R., D. Solo, M. Bartel, J. Boyer , B. Fox , E. Simon. *XML-Signature Syntax and Processing*, World Wide Web Consortium. http://www.w3.org/TR/xmldsig-core/

[XML-Schema1] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn. *XML Schema Part 1: Structures*, W3C Working Draft 22 September 2000, http://www.w3.org/TR/xmlschema-1/

[XML-Schema2] P. V. Biron, A. Malhotra, *XML Schema Part 2: Datatypes*; W3C Working Draft 22 September 2000, http://www.w3.org/TR/xmlschema-2/

[X.509] rfc2253 Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names

Appendix D Legal Notices

Copyright

Intellectual Property Statement

Disclaimer