Security Services Markup Language

Prateek Mishra	*Netegrity*
Phillip Hallam-Baker	*VeriSign*
Zahid Ahmed	*Commerce One*
Alex Ceponkus	*Bowstreet*
Marc Chanliau	*Netegrity*
Jeremy Epstein	*webMethods*
Chris Ferris	*Sun Microsystems*
David Jablon	*Netegrity*
Eve Maler	*Sun Microsystems*
David Orchard	*Jamcracker*

Reviewers:	*ATG, Inc*
	TIBCO
	Oracle
	PWC

01/08/01

Draft Version 0.8a: January 8, 2001

Security Services Markup Language

Version 0.8a

1 EXECUTIVE SUMMARY 2

2 REQUIREMENTS 3

2.1 Leveraging Existing Technology 3

2.2 Foundations 3

2.3 Services 4

2.4 Design Goals 4

2.5 Scope and Limitations 5

3 S2ML USE CASE SCENARIOS 5

3.1 Scenario #1: User-Driven Transactions (Single Sign-On) 5

3.2 Scenario #2: Service-Driven Transactions (Trade Exchange) 7

3.3 Scenario #3: Hosted Services (Security ASP) 8

4 ARCHITECTURE 9

4.1 Name Assertions and Entitlements 10

4.2 Secret Name Assertions vs. Scoped Name Assertions 12

4.2.1 Scoped Name Assertions 12

4.2.2 Secret Name Assertions 12

4.3 Authentication (Auth) and Authorization (Az) Services 13

4.4 Assertion Validity 17

4.4.1 Audience Restriction 18

5 MESSAGE SET 18

5.1 URI Naming Infrastructure 18

5.2 Common Structures 18

5.2.1 ID Element 19
5.2.2 Issuer Element 19
5.2.3 ValidityInterval Complex Type 19
5.2.4 Date Element 20
5.2.5 Audiences Element 20
5.2.6 DependsOn Element 20
5.2.7 InResponseTo Element 20
5.2.8 Holder Element 20
5.2.9 ResultCode Simple Type 20

5.3 Authentication Messages 21
5.3.1 AuthRequest Element 21
5.3.2 AuthResponse Element 21
5.3.3 Credentials Element 21
5.3.4 NameAssertion Element 22
5.3.5 AuthData Element 23

5.4 Authorization Messages 24
5.4.1 AzRequest Element 24
5.4.2 AzResponse Element 25
5.4.3 Entitlement Element 26

6 BINDINGS TO MESSAGING AND TRANSPORT PROTOCOLS 27

6.1 Web Browser Binding 27

6.2 HTTP Binding 28

6.3 MIME Binding 29
6.3.1 Example of S2ML Security Package 30
6.3.2 Example of Encrypted S2ML Security Package 30
6.3.3 Example of S2ML Security Package Combined with Business Payload 30
6.3.4 Example of Clear Signed MIME Package 31

6.4 ebXML Binding 31

6.5 SOAP Binding 31

7 CONFORMANCE 32

APPENDIX A URI EQUALITY: LEXICAL COMPARISON 33

APPENDIX B: S2ML SCHEMA 35

APPENDIX C REFERENCES 39

APPENDIX D LEGAL NOTICES 40

1 Executive Summary

This specification defines Security Services Markup Language (S2ML), a protocol for two security services: authentication and authorization. The protocol consists of request and response pairs of XML documents for each service. This specification provides a schema that governs these XML documents, as well as bindings to several message and transport protocols with which S2ML might be used.

S2ML recognizes that there are a wide range of authentication technologies in use, such as login-password, SSL, Digital Signing, Kerberos, and Smart Cards. There are also many frameworks for authorization, including ACLs, Capabilities, and the Java Authorization Model. 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, which can convey the results established by a wide variety of existing security mechanisms

S2ML defines two key XML elements—*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 by document flow, may involve cooperating but distinct enterprises. Transactions may originate at a workstation, and with the help of a portal or marketplace site, pass through a series of staged interactions with other sites. For example, one site may authenticate a name-to-credential binding while another site provides additional assessment of the named user's capabilities to perform a transaction. 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 elements and security interfaces are described in this document:

- *NameAssertion:* the result of successful authentication is a digitally signed XML assertion describing the authentication type, subject name 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 *AuthRequest* element contains *credentials*; the result of authentication is an *AuthResponse* element containing a *NameAssertion* and possibly also *Entitlements.*

- *Authorization:* An *AzRequest* element contains a *NameAssertion*, zero or more *Entitlements*, and an authorization *Question*; the result of authorization is an *AzResponse* element that contains an *Answer* and may also include *Entitlements*.

Auditing, based on logging and analysis of security-related data, is a key requirement in security systems. S2ML supports auditing by including information in requests and responses that may be used to establish their sequencing relationships over long time periods.

2 Requirements

Following are the requirements that underlie S2ML.

2.1 Leveraging Existing Technology

A central requirement for S2ML is to build on existing security technologies, including those viewed as standard web authentication technologies. These include:

- Server-authenticated SSL connections from browser to web server
- Password and user-certificate authentication from web browser
- Existing web server and related user authentication mechanisms
- Existing secure server-to-server programming infrastructure based on SSL, S/MIME, and XML Signature [XML-SIG].

2.2 Foundations

This work provides a standardized way to expand the foundation for secure services, including securely authenticated message flows or transactions initiated by end-users or services, using private or secret keys or passwords, for authentication.

The following features are required:

- A standard representation of a subject's authentication "event" which is (1) extensible and (2) cryptographically bound to an issuer (Name Assertion).
- A standard representation of an authenticated subject's properties which is (1) extensible and (2) cryptographically bound to an issuer (Entitlement Assertion).
- Ability to "bind" Name and Entitlement Assertions into standard Internet protocols. Should include single and multi-step message flows based on:
  (1) standard commercial browsers,
  (2) HTTP as a transport protocol ,
  (3) MIME as a packaging protocol,
  (4) SOAP as a messaging protocol,
  (5) ebXML as a messaging protocol.
- Support for securing message flows or transactions that involve separately administered security engines; said security engines may be manufactured by many different vendors.
- Support for "pushing" name and entitlement assertions from one zone of security administration to another; support for "pulling" name and entitlement assertions from one zone of security administration to another.

2.3 Services

The required features may be used to provide authentication and authorization services.

Required features of an authentication service include:

- The ability to certify the occurrence of authentication "events"
- The description of the certification request and response in XML
- The ability to authenticate based on name-password, public and private keys, and certificates

Required features of an authorization service include:

- The description of the authorization request and response in XML
- Extensibility to handle a broad class of authorization models
- Authorization requests and responses that are "scoped" by name and entitlement assertions carried with the message
- Authorization requests and responses that are able to make explicit references to name and entitlement assertions

All services require:

- Support for clear separation between security policies and interfaces utilized to access those policies

2.4 Design Goals

S2ML should:

- Be based on [XML1.0], [XML Schema], [XML-SIG], and (emergent) [XML-Encryption] specifications
- Complement related efforts such as [XKMS] and any others that emerge
- Have a layered architecture distinguishing clearly between pre-existing infrastructure, new foundations, and more derived concepts

2.5 Scope and Limitations

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

Non-repudiation services and markup are outside the scope of S2ML 1.0.

Native 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.

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 (Single Sign-On)

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 user 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 as part of its trusted relationship.

In user-driven transaction environments, name assertions can "travel" with the user in various ways, typically using the URL query string, HTTP headers, or cookies.

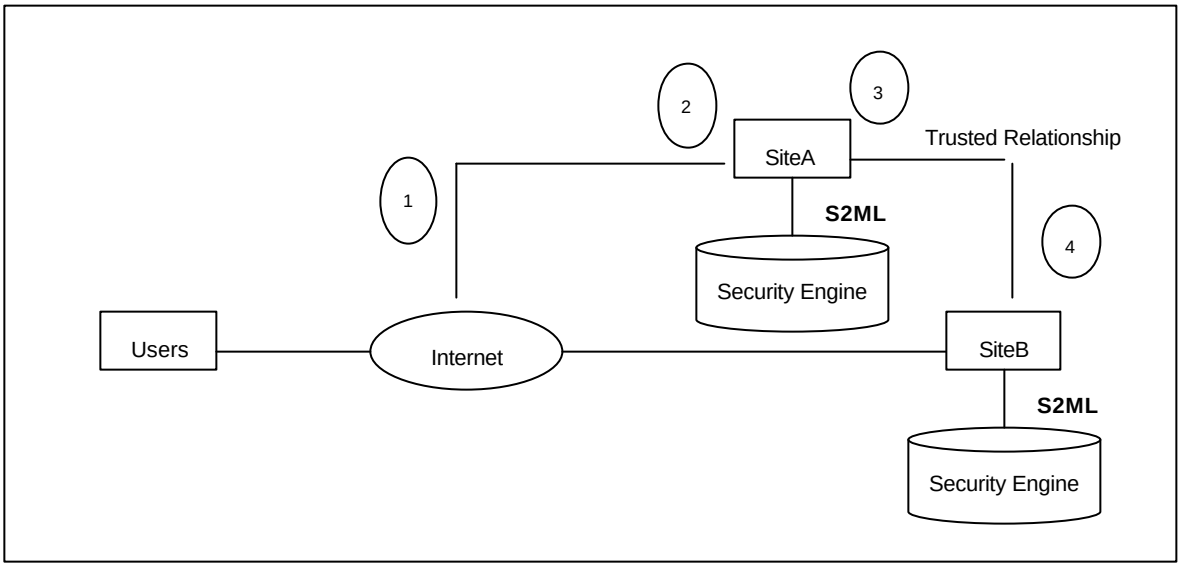
In the following example, users are able to visit SiteA and SiteB seamlessly.

(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 name assertion and one or more entitlements. S2ML assertions may be generated through the use of an 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.

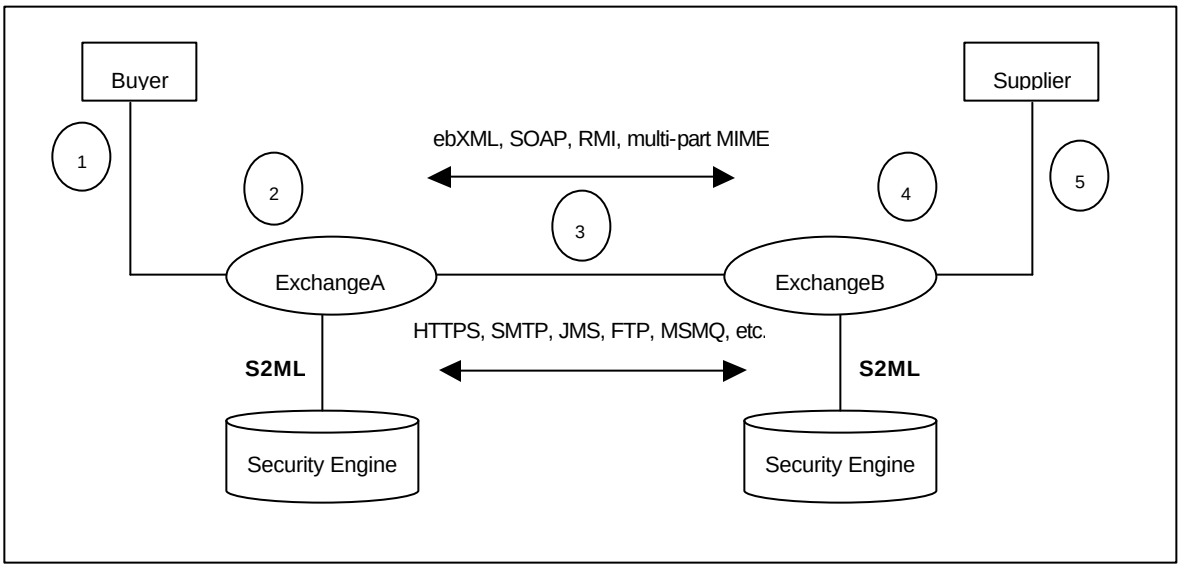
(4) User is allowed into SiteB without having to re-authenticate as information about the S2ML name assertion travels with the user on the URL line. Site B must check the received name assertion for validity within the context of the relationship between the two sites. The information about the user authenticated at SiteA together with the entitlements from SiteA may be used to complete the transaction at SiteB. SiteB may also further query SiteA about the user's authorizations. Thus authorization information may be both "pushed" from A to B in the form of assertions bound to the user's transactions, and also "pulled" on an as-needed basis by B from A.



3.2 Scenario #2: Service-Driven Transactions (Trade Exchange)

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 name and 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, such as a user name and password. (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 may optionally remove the credentials from the document.

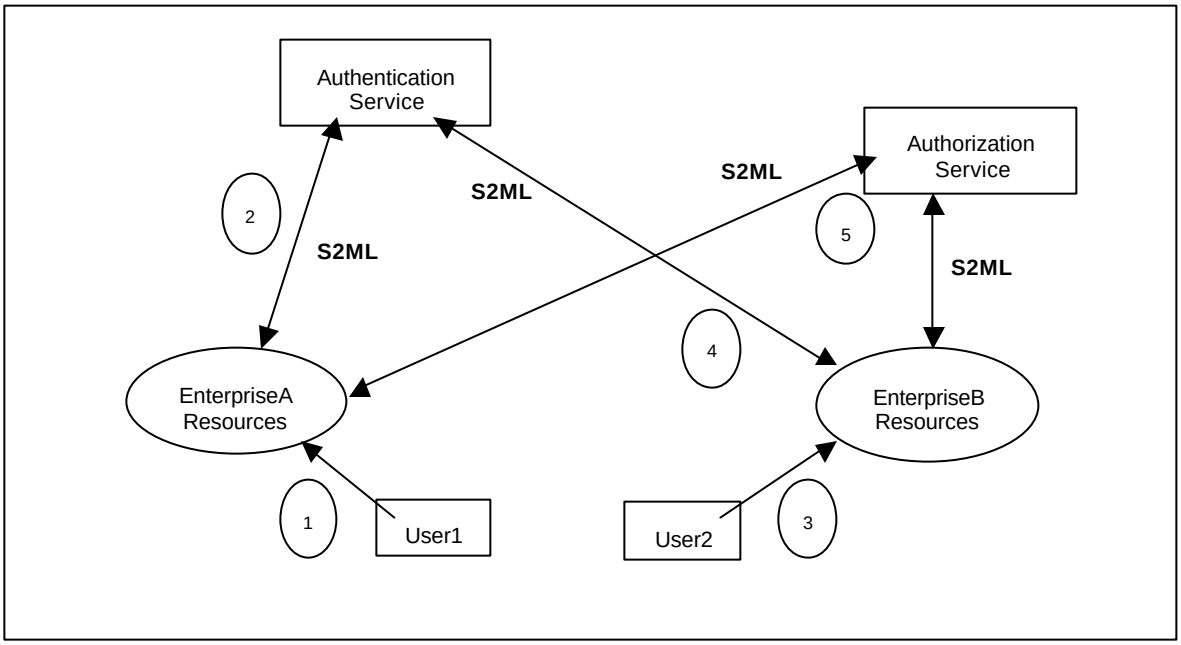
(3) The message is sent to ExchangeB using any messaging framework (SOAP, ebXML, RMI, multi-part MIME, RosettaNet, etc.) over any transport protocol (HTTPS, SMTP, JMS, FTP, MSMQ, IBMMQ, etc.)

(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 (Security ASP)

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. 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).

User1 and User2 access services (business processes) at EnterpriseA and EnterpriseB that require authorization (for example, to make a purchase). The authorization process for both users utilizes the same remote authorization service.

4 Architecture

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 [RFC2119].

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, or structure of actors and security services; the only restriction placed is that each actor MUST have a unique name (URI) [RFC2396]. 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, or SMTP. 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 usable 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*. The creation and management of the trust relationship is outside the scope of S2ML. In addition, depending on 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-SIG] specification. It is important to note that [XML-SIG] supports both using secret-key (for example, HMAC) and public-key signing. When the XML encryption specifications are available, additional infrastructure 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-SIG]. 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[X-TASS] 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 the issuer's identity to attributes of the assertion

A Name Assertion describes a successful authentication step:

```
<NameAssertion>
    <ID>urn:authEngine32:xsde12</ID>
    <Issuer>http://www.example.com/authEngine32</Issuer>
    <Date>2000-10–16T12:34:120-05:00</Date>
    <Audiences>urn:all_example_servers</Audiences>
    <AuthData>
         <AuthType>Login</AuthType>
      <IdentityToken>x12+21defqa$3#</IdentityToken>
    </AuthData>
    <dsig:Signature>. . . </dsig:Signature>
</NameAssertion>
```

The name assertion above indicates that the actor with the identifying URI

`http://www.example.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

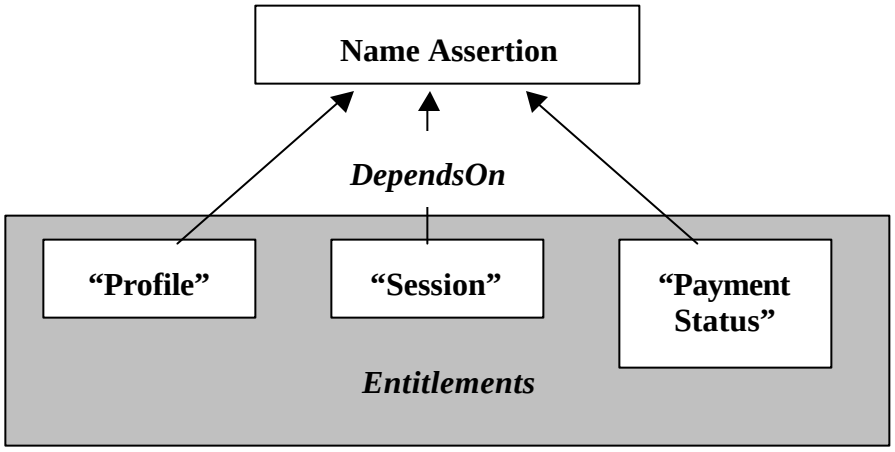
<AuthData> provide details about the authentication act: in this case, the subject provided a password and user name, and the issuer has provided an identity token.

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>
    <ID>urn:financeDepartment:129de12</ID>
    <Issuer>http://www.example.com/finance/AzEngine</Issuer>
    <Date>2000-10–16T12:34:120-05:00</Date>
    <Audiences>urn:all_example_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 transaction's 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 Secret Name Assertions vs. Scoped Name Assertions

This section describes two security models for the use of Name Assertions.

In the first model the Name Assertion is a secret value that must be secured with other mechanisms to prevent vulnerability to theft and misuse through impersonation. In the second model, the Name Assertion can be a public value without destroying its utility.We distinguish these models by differentiating between Secret Name Assertions and Scoped Name Assertions, where each has a distinct but related syntax. The following describes these in more detail.

4.2.1 Scoped Name Assertions

In this situation, a name assertion is cryptographically bound to a specific payload through a digital signature. The `Holder` element is introduced to provide within Name Assertions the public key of the actor binding the name assertion to a payload using a signature.

A scoped name assertion MUST be combined with a payload and the resulting composite object MUST be signed by the actor's private key. The specific packaging and signing details are described in the bindings section; this currently includes MIME, SOAP, and ebXML.

A relying party MUST validate the signature of the composite object carrying the name assertion using the public key identified in the Holder element of the NameAssertion. This will ensure that the name assertion is used within the scope for which it was designated by its original holder.

Two patterns of use are noted for scoped name assertions. In the first, a subject (typically an end-user) *without* a private/public key pair presents credentials for authentication to a server together with a payload for further processing. In this case, only the server can bind the resulting name assertion to the payload by use of the server private key and must include the server public key in the generated name assertion `Holder` element. In this case, there is an assumption that the server is trusted and can act on behalf of the user.

In the second, a subject presents credentials to a server and receives a name assertion which includes the *subject* public key in the returned name assertion `Holder` element. The subject may now bind the name assertion to a payload and submit the signed package directly to other servers. In this case, there is no need to make an assumption that the server will act on behalf of the user.

4.2.2 Secret Name Assertions

In this situation, Name Assertions are not cryptographically bound to a specific payload and essentially act as "bearer" tokens. The Holder element is absent from the name assertion. This type of name assertion MUST be kept secret and used only over encrypted transports such as SSL. It is also recommended that the lifetime of such name assertions (using the ValidityInterval element) be kept to the absolute minimum.

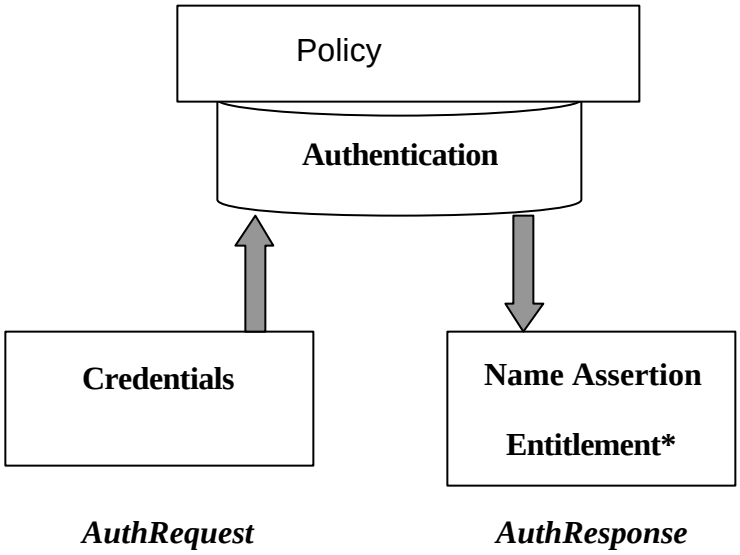
A typical use case for secret name assertions is when a user (with no private-public key pair) at a

workstation logs on to a web site. In such a case, a name assertion may be bound to the user, either as a cookie or as part of a URL line.

The S2ML specification also describes a notion of name assertion *reference*. Exactly, the same considerations apply to references: If the reference is not cryptographically bound to a payload it MUST be kept secret and used only over encrypted transports such as SSL.

4.3 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 certification service which validates subject credentials and, if successful, returns a name assertion and zero or more entitlements appropriate to the subject.

For the case when the authentication service is invoked with public key credentials, it MUST be used to complement existing authentication protocols based on SSL or document signature verification (XML-SIG, S/MIME). The intent here is to separate out a policy-driven component of authentication into a separate service which can be shared, for example, by all servers in a web farm. It should not be viewed as a replacement for existing authentication protocols.

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, obtained from the transport layer, etc.

```
<AuthRequest>
    <ID>urn:JavaServletPlugInRequest:988</ID>
    <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: no credentials, login, X509 certificates, and public keys. The `Credentials` element also permits the use of foreign namespaces. This may be used as the means for extension to other authentication schemes.

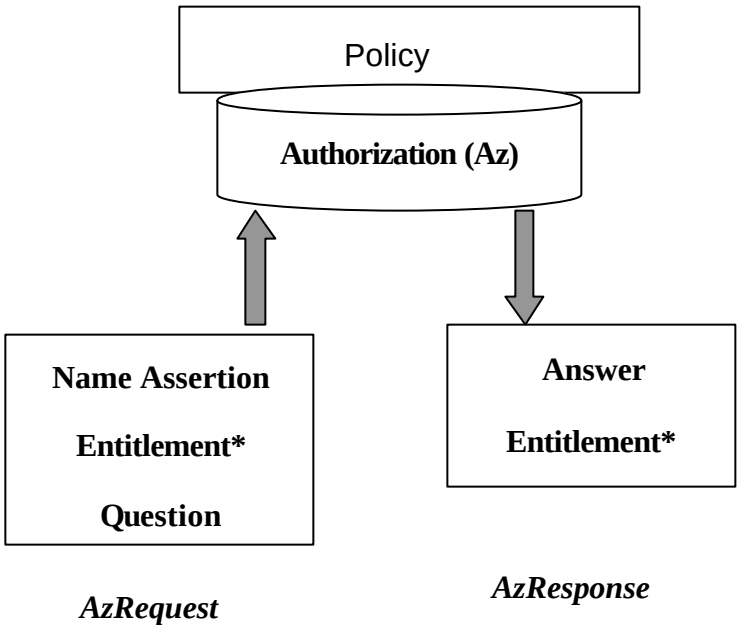
The authentication engine responds with an `AuthResponse` message; if authentication succeeds, the message includes a name assertion describing the authentication type and subject attributes.

```
<AuthResponse>
    <ID>urn:MainAuthServer:0981</ID>
    <Date>2000-11–16T12:34:120-05:00</Date>
    <InResponseTo>urn:JavaServletPlugInRequest:988</InResponseTo>
    <Result>Success</Result>
    <NameAssertion>
        <ID>urn:authEngine32:xsde12</ID>
        <Issuer>http://www.example.com/authEngine32</Issuer>
        <Date>2000-11–16T12:36:120-05:00</Date>
        <Audiences>urn:all_example_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 which may be contained within an `AuthData` element:

- `UserHandle` element, consisting of a string user-store name and an X.509 distinguished name string
- `IdentityToken` element, consisting of a string
- X509 Certificate
- Public keys

The `AuthData` element also permits the use of foreign namespaces. 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 an `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 authorization model, `SimpleAz`, and gives it the URI `http://az.s2ml.org/SimpleAz`.

This model describes a class of authorization questions of the form "VERB Resource" (for example, `GET http://www.example.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 `UnknownAzModel` 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 MUST implement the `SimpleAz` model in addition to any other implemented models.

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

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

4.4 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. The issuer's digital signature is valid at the time of scrutiny and binds to required elements in the assertion,
3. The time period for which the assertion is being scrutinized lies within the time period specified by the `ValidityInterval` element, and
4. The business relationship between the actor and issuer references at least one of the `Audience` elements. The creation and management of business relationships is outside the scope of the specification.

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

4.4.1 Audience Restriction

Assertions MAY be addressed to a specific audience. Although an actor 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. An audience is identified by a URI that identifies a document that describes the terms and conditions of audience membership. Examples of terms and conditions include:

- 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

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_rulebook`
: Client accepts the provisions of the *bizexchange* rulebook.

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.

5 Message Set

This section details the meaning and usage of the XML constructs that make up S2ML messages. The full schema is provided in Appendix B.

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 of the use of a URI as an object *locator*. S2ML introduces objects such as audiences and authorization roles that can be identified for purposes of comparison even though there is no means of locating or even resolving them. Appendix A describes comparison rules for determining URI equality for S2ML purposes.

5.2 Common Structures

The following elements are used in various locations in S2ML messages.

5.2.1 ID Element

The `ID` element specifies a label by means of a URI.

```
<xsd:element name="ID" type="xsd:uriReference"/>
```

5.2.2 Issuer Element

The `Issuer` element specifies the issuer of the assertion by means of a URI.

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

5.2.3 ValidityInterval Complex Type

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

```
<xsd:complexType name="ValidityInterval">
  <xsd:sequence>
    <xsd:element name="NotBefore" type="xsd:timeInstant" minOccurs="0"/>
    <xsd:element name="NotAfter" type="xsd:timeInstant" minOccurs="0"/>
  </xsd:sequence>
</xsd: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.

For purposes of comparison the time interval `NotBefore` to `NotAfter` begins at the earliest time instant compatible with the specification of `NotBefore` and ends 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`, `12:05:11.9999`, and `12:05:12.00` are within the time interval. The time `12:05:12.0001` is outside the time interval.

5.2.4 Date Element

The `Date` element MUST fully specify the date.

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

5.2.5 Audiences Element

The `Audiences` element specifies a set of audiences to which the assertion is addressed.

```
<xsd:simpleType name="listOfUriRefs">
  <xsd:list itemType="xsd:uriReference"/>
</xsd:simpleType>

<xsd:element name="Audiences" type="listOfUriRefs"/>
```

5.2.6 DependsOn Element

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

```
<xsd:element name="DependsOn" type="xsd:uriReference"/>
```

5.2.7 InResponseTo Element

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

```
<xsd:element name="InResponseTo" type="xsd:uriReference"/>
```

5.2.8 Holder Element

The `Holder` element describes a public key value or X509Data.

```
<xsd:element name="Holder" minoccurs="0"/>
          <xsd:complexType>
            <xsd:element ref="dsig:X509Data"/>
            <xsd:element ref="dsig:KeyValue"/>
          </xsd:complexType>
```

5.2.9 ResultCode Simple Type

The `ResultCode` enumerated type 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.

UnknownAzModel

The authorization engine is does not support the AzModel used in the request.

NotEnoughInfo

The service lacks information required to respond definitively to the request.

```
<xsd:simpleType name="ResultCode" base="xsd:string">
  <xsd:enumeration value="Success"/>
  <xsd:enumeration value="Failure"/>
  <xsd:enumeration value="UnknownAzModel"/>
  <xsd:enumeration value="NotEnoughInfo"/>
</xsd:simpleType>
```

5.3 Authentication Messages

This section describes the structures for authentication-related messages.

5.3.1 AuthRequest Element

The `AuthRequest` element contains an authentication request message.

```
<xsd:element name="AuthRequest">
  <xsd:complexType>
    <xsd:all>
      <xsd:element ref="ID"/>
      <xsd:element ref="Time"/>
      <xsd:element ref="Holder" minoccurs="0"/>
      <xsd:element ref="Credentials"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

5.3.2 AuthResponse Element

The `AuthResponse` element contains an authentication response message.

```
<xsd:element name="AuthResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ID"/>
      <xsd:element ref="Time"/>
      <xsd:element ref="InResponseTo"/>
      <xsd:element name="Result" type="ResultCode"/>
      <xsd:element ref="NameAssertion" minOccurs="0"/>
      <xsd:element ref="Entitlement" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

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

5.3.3 Credentials Element

The `Credentials` element contains one of the following:

- A `Login` element for providing user name-password information
- An `X509Data` element (from [XML-SIG])
- A `KeyValue` element (from [XML-SIG])
- A `NoCredentials` element to signify that no credentials are being provided
- An element that is in a namespace other than S2ML

```
<xsd:element name="Credentials">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref ="Login"/>
      <xsd:element ref="dsig:X509Data"/>
      <xsd:element ref="dsig:KeyValue"/>
      <xsd:any namespace="##other"/>
      <xsd:element name="NoCredentials"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

5.3.3.1 Login Element

The `Login` element must contain a name and password pair; it may also contain an optional `Domain` element.

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

5.3.4 NameAssertion Element

The `NameAssertion` element contains all the information related to a name assertion.

```
<xsd:element name="NameAssertion">
  <xsd:complexType>
    <xsd:all>
      <xsd:element ref=“Holder” minoccurs=”0”/>
      <xsd:element ref="ID"/>
      <xsd:element ref="Issuer"/>
      <xsd:element ref="Date"/>
      <xsd:element ref="Audiences" minOccurs="0"/>
      <xsd:element ref="ValidityInterval" minOccurs="0"/>
      <xsd:element ref="AuthData"/>
      <xsd:element ref="dsig:Signature"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

The subelements for `NameAssertion` have the following meaning and usage:

Identifier	Type	Description
`Holder`		Optional; describes public key used to validate a scoped name assertion binding to a payload.
`ID`	`URI`	MUST be present and be unique over all Name Assertions.
`Issuer`	`URI`	MUST be present.
`Date`	`timeInstant`	MUST be present
`AuthData`		Information generated by authentication step; MUST be present.
`ValidityInterval`		Optional.
`Audiences`		Optional.
`dsig:Signature`		MUST be present; enveloped digital signature binding the issuer's identity to required assertion attributes.

5.3.5 AuthData Element

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`, or `dsig:KeyValue`, or are provided using XML structures from a namespace other than S2ML

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

```
</xsd:element>
```

5.3.5.1 AuthType Element

The `AuthType` element identifies the type of authentication data that is to follow in this `AuthData` element.

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

5.3.5.2 UserHandle Element

The `UserHandle` element represents the case wherein credentials are mapped to an entry within a directory or user store. The contents of 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`

```
<xsd:element name="UserHandle">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="Directory" type="xsd:string"/>
      <xsd:element name="X509.DN" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

5.3.5.3 IdentityToken Element

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

5.4 Authorization Messages

This section describes the structures for authorization-related messages.

5.4.1 AzRequest Element

The `AzRequest` element contains an authorization request message.

```
<xsd:element name="AzRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref ="ID"/>
      <xsd:element ref ="Time"/>
      <xsd:element ref ="NameAssertion"/>
```

```
        <xsd:element ref ="Question"/>
        <xsd:element ref ="Entitlement" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

5.4.1.1 Question Element

```
  <xsd:element name="Question">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element ref="ResourceContext"/>
        <xsd:any namespace="##other"/>
      </xsd:choice>
      <xsd:attribute name="AzModel" type="xsd:uriReference"/>
    </xsd:complexType>
  </xsd:element>
```

5.4.1.2 ResourceContext Element

The `ResourceContext` element provides information about the resource context.

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

The sub-elements for `ResourceContext` have the following meaning and usage:

Identifier	Type	Description
`Resource`	`URI`	The resource name. MUST be present.
`Method`	`URI`	Verb. MUST be present.

5.4.2 AzResponse Element

The `AzResponse` element contains an authorization response message.

```
  <xsd:element name="AzResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="ID"/>
        <xsd:element ref="Time"/>
        <xsd:element ref="InResponseTo" />
        <xsd:element name ref="Answer"/>
        <xsd:element name ref="entitlement" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

5.4.2.1 Answer Element

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

5.4.3 Entitlement Element

The `Entitlement` element contains all the information related to an entitlement assertion.

```
<xsd:element name="Entitlement">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name ref="ID"/>
      <xsd:element name ref="Issuer"/>
      <xsd:element name ref="Date"/>
      <xsd:element name ref="Audiences" minOccurs="0"/>
      <xsd:element name ref="DependsOn"/>
      <xsd:element name ref="AzData"/>
      <xsd:element name="ValidityInterval" type="ValidityInterval" minOccurs="0"/>
      <xsd:element name ref="dsig:Signature"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

The sub-elements for `Entitlement` have the following meaning and usage:

Identifier	Type	Description
`ID`	`URI`	MUST be present and be unique over all entitlements.
`Date`	`timeInstant`	MUST be present.
`Issuer`	`uriRef`	MUST be present.
`DependsOn`	`uriRef`	Link to Name Assertion; MUST be present.
`AzData`		MUST be present.
`ValidityInterval`	`ValidityInterval`	Optional.
`Audiences`		Optional.
`dsig:Signature`		MUST be present; enveloped digital signature binding the issuer's identity to assertion attributes.

5.4.3.1 AzData Element

The `AzData` element provides data about the authorization.

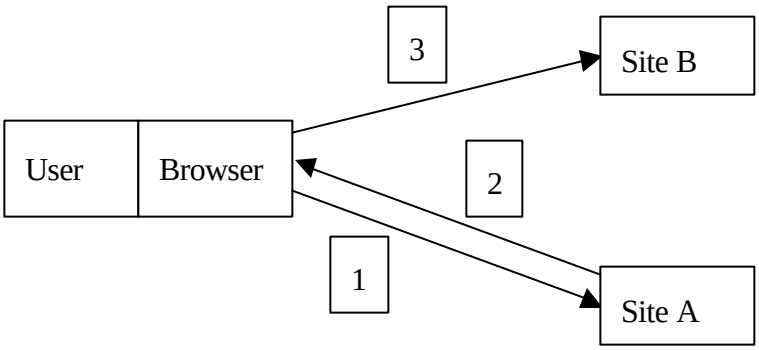
```
<xsd:element name="AzData">
  <xsd:complexType>
    <xsd:all>
      <xsd:any namespace="##other"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

6 Bindings to Messaging and Transport Protocols

It is a goal of S2ML to be neutral with respect to messaging and transport protocols, such that bindings could be created to any one protocol. Bindings to several common protocols are provided in this section.

6.1 Web Browser Binding

In many user-driven scenarios there is a need to communicate security information through cookies or URL query strings. In such cases, assertions originating from site A may need to be communicated to site B via the user's browser.



As S2ML assertions may be of variable size and as both URLs and cookies are strongly size constrained, this specification describes a system in which *unambiguous references* to S2ML assertions are conveyed through short fixed-size strings. Using such references, sites may retrieve S2ML assertions from other sites through means that lie outside the scope of this specification.

The separate notions of References and Secret References are defined here. Secret References must be used when the Assertion referred to is a Secret Assertion. A Secret Reference may be used as a "bearer" token in place of the Secret Assertion itself.

Note that a Reference itself is not a signed object. It is assumed that the authenticity and integrity of a Reference may be verified in the act of retrieving the associated Assertion.

A Reference refers to a *single* assertion, and is constructed from two components:

Sender Identifier	uniquely identifies the Sender URI
Assertion Identifier	uniquely identifies the contents of `This` element of an Assertion.

Identifiers should be guaranteed to be unique within the scope of use. For the case of an Assertion Identifier in a Secret Reference, it is essential that the size be appropriately large to prevent forgery of valid Assertion Identifiers. We recommend that any Assertion Identifiers be constructed from an appropriately sized cryptographic hash function, to prevent possible collisions between an artificially constructed Assertion Identifier and the set of valid Secret Assertions.

The S2MLreference conforms to the standard HTTP header format as in [RFC2068], with the name field set to "S2MLreference" and the value field containing the Reference.

```
<S2Mlreference> :=       “S2Mlreference:” <Reference>
<Reference> :=           “v1” <SenderIdentifier> “:” <AssertionIdentifier>
<SenderIdentifier> :=    B64 representation of SHA1 hash of Sender URI
<AssertionIdentifier> := B64 representation of SHA1 hash of Assertion <ID> element
```

The construction of the SenderIdentifier and AssertionIdentifier uses a B64 representation, as specified in [MIME], of SHA1 hash results of a Sender URI and the <ID> element of the associated Assertion, respectively.

The remainder of this section gives suggestions for implementation and use of these References.

An originating site may store and manage a table of assertions for which references have been exported outside the site. The table would be indexed by the Assertion Identifier element of each Reference, which is constructed from a hash function of the referenced Assertion. The receiving site would maintain a table of Sender Identifiers and sending site descriptions. The receiving site would decode and verify the Assertion Identifier and Sender Identifier, determine the sending site from the Sender Identifier, and contact the sending site to retrieve the relevant Assertion associated with the Assertion Identifier.

6.2 HTTP Binding

For this case, we are interested in conveying assertions from an originating site to a receiving site, where the originating site is sending an HTTP command to the receiving site. Two methods are defined:

(1) Use of HTTP headers to convey assertion references in the format presented in 6.1,

(2) Incorporation of S2ML assertions within the body of an HTTP POST command. Details are TBD.

6.3 MIME Binding

MIME and particularly Multipart-MIME are very commonly used in messaging systems. In this section, we describe the use of MIME as a packaging technique for combining one or more S2ML documents into a single MIME entity, a *security package*. We also describe the use of S/MIME to encrypt security packages and to *bind* a security package to a business payload.

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. Individual messaging frameworks may provide alternative ways to include S2ML fragments and assertions with messages.

S2ML documents such as Credentials, Name Assertions, and Entitlements MUST be packaged using MIME *multipart/mixed* into a single MIME security package. Each S2ML document is packaged as a single MIME object with content-type *text/xml*. The security package MAY be optionally encrypted with S/MIME enveloping using *application/pkcs7-mime* content-type. A credential document that includes login password information may need to be protected over non-secure transports. Entitlement information requiring confidentiality may also need to be protected if it is being processed over multiple sites over varying transport protocols.

The security package MUST be combined with the message payload using content-type *multipart/related* with the business payload comprising the root of the message.

Finally, the message payload and the S2ML security package MUST be bound together. This is achieved either by using SSL with a client-side certificate or by signing. If signing is used, the document will have content-type *multipart/signed*. The purpose of the signature is to ensure that the security package combined with the original message payload such that no intermediary party can replace the original S2ML security package (e.g., during message transmissions) with alternatives. This signature block is optional and some messaging frameworks may choose to rely on transport-level security (e.g., SSL). The act of binding the security package to a message payload also serves to scope any name assertions contained within the security package (Section 4.2).

Note: if an S2ML payload will be transferred over a transfer medium that is constrained to 7-bit text (e.g., parts of the Internet SMTP infrastructure), then the charset attribute of the Content-Type header should be set to "us-ascii", or another 7-bit character set. However, specifying a value for the Content-Transfer-Encoding header can relax this requirement. Using the "quoted-printable" value of the Content-Transfer-Encoding header, for instance, allows an 8-bit character set (such as iso-8859-1 or UTF-8) to be sent over a 7-bit network. Binary data can have a Content-Transfer-Encoding type of "base64" to solve the 7-bit issue. Note that 8-bit clean transports do not encode 8-bit character sets, but still may require base64 encoding for binary data. So-called "binary clean" transport protocols (such as HTTP) do not require encoding.

6.3.1 Example of S2ML Security Package

```
   MIME-Version: 1.0
      Content-Type:multipart/mixed;
      boundary="boundary1"

       --boundary1

       Content-Type:text/xml;
       charset=us-ascii

       <NameAssertion>...</NameAssertion>

       --boundary1

       Content-Type:text/xml;
       charset=iso-8859-1
       Content-Transfer-Encoding:quoted-printable

        <Entitlement>...</Entitlement>

        --boundary1--
```

6.3.2 Example of Encrypted S2ML Security Package

Encrypted multipart MIME documents without signing are packaged as type *application/pkcs7-mime*. The example below assumes that a multipart MIME document (such as listed above) has already been created.

```
              MIME Version: 1.0
           Content-Type:application/pkcs7-mime;
           smime-type=envelope-data;
           name=smime.p7m
           Content-Transfer-Encoding:base64
           Content-Disposition:attachment;
           filename=smime7.p7m

[Encrypted S2ML security package, whose MIME type is multipart/mixed when decrypted]
```

6.3.3 Example of S2ML Security Package Combined with Business Payload

```
   MIME-Version: 1.0
   Content-Type: Multipart/Related;
   boundary=”payload+security”
   --“payload+security”
   Business Payload
   --“payload+security”
   Security Package
   --“payload+security”
```

6.3.4 Example of Clear Signed MIME Package

```
    MIME-Version: 1.0
Content-Type:multipart/signed;
protocol="application/pkcs7-signature";
micalg=sha1;
boundary="boundary2"

    --boundary2

    Content-Type: Multipart/Related;
    boundary=”payload+security”
    --“payload+security”
    Business Payload
    --“payload+security”
    Security Package
    --“payload+security”

    --boundary2

Content-Type:application/pkcs7-signature;
name=smime2.p7s
Content-Transfer-Encoding:base64
Content-Disposition:attachment;
filename=smime2.p7s
     [SIGNATURE GOES HERE]

    --boundary2--
```

6.4 ebXML Binding

TBD

6.5 SOAP Binding

Binding Notes:
- It is up to the application to decide whether the 'mustUnderstand' attribute is applied to the headers.
- It is up to the application to decide whether the 'Actor' attribute is 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 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 if it provides authorization decisions based on name assertions and entitlements generated elsewhere.

2. A security service which is an 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-producer** of S2ML if it can both produce and consume name assertions and entitlements.

4. A security system which is an S2ML consumer-producer may also provide an 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 is 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.
- Only absolute URIs are utilized within S2ML.

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 not 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/
```

[ISSUE: These definitions are not complete; for example, what about hierarchical URIs?]

Appendix B: S2ML Schema

This appendix describes the XML schema for S2ML. The namespace URI identifying the element set described in this document is `http://ns.s2ml.org/s2ml`.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema
        targetNamespace=”http://ns.s2ml.org/s2ml”
        xmlns:dsig = "http://www.w3.org/2000/9/xmldsig#"
         xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema">
  <xsd:import namespace = "http://www.w3.org/2000/9/xmldsig#"
       schemaLocation = "http://www.w3.org/2000/9/xmldsig%23"/>
  <xsd:annotation>
     <xsd:documentation>
      ============================================================
        This document is the XML schema for the S2ML specification V0.8a
      Latest Update: January 06, 2001
      ============================================================
      Conforms to w3c http://www.w3.org/2000/10/XMLSchema
      ============================================================
        URI identifying the element set described in this document:
      http://ns.s2ml.org/s2ml
      ============================================================
                 </xsd:documentation>
  </xsd:annotation>
  <xsd:element name = "ID" type = "xsd:uriReference"/>
  <xsd:element name = "Issuer" type = "xsd:uriReference"/>
  <xsd:complexType name = "ValidityInterval">
     <xsd:sequence>
        <xsd:element name = "NotBefore" type = "xsd:timeInstant" minOccurs = "0"/>
        <xsd:element name = "NotAfter" type = "xsd:timeInstant" minOccurs = "0"/>
     </xsd:sequence>
  </xsd:complexType>
  <xsd:element name = "Date" type = "xsd:timeInstant"/>
  <xsd:simpleType name = "listOfUriRefs">
     <xsd:list itemType = "xsd:uriReference"/>
  </xsd:simpleType>
  <xsd:simpleType name = "ResultCode">
     <xsd:restriction base = "xsd:string"/>
  </xsd:simpleType>
  <xsd:element name = "Audiences" type = "listOfUriRefs"/>
  <xsd:element name = "DependsOn" type = "xsd:uriReference"/>
  <xsd:element name = "InResponseTo" type = "xsd:uriReference"/>
  <xsd:element name = "NameAssertion">
     <xsd:complexType>
        <xsd:all>
           <xsd:element name = "Holder"/>
           <xsd:element ref = "ID"/>
           <xsd:element ref = "Issuer"/>
           <xsd:element ref = "Date"/>
           <xsd:element ref = "Audiences" minOccurs = "0"/>
```

```
            <xsd:element ref = "ValidityInterval" minOccurs = "0"/>
            <xsd:element ref = "AuthData"/>
            <xsd:element ref = "dsig:Signature"/>
        </xsd:all>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "AuthRequest">
    <xsd:complexType>
        <xsd:all>
            <xsd:element ref = "ID"/>
            <xsd:element ref = "Time"/>
            <xsd:element ref = "Credentials"/>
        </xsd:all>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "AuthResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref = "ID"/>
            <xsd:element ref = "Time"/>
            <xsd:element ref = "InResponseTo"/>
            <xsd:element name = "Result" type = "ResultCode"/>
            <xsd:element ref = "NameAssertion" minOccurs = "0"/>
            <xsd:element ref = "Entitlement" minOccurs = "0" maxOccurs = "unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "Login">
    <xsd:complexType>
        <xsd:all>
            <xsd:element name = "Name" type = "xsd:string"/>
            <xsd:element name = "Password" type = "xsd:string"/>
            <xsd:element name = "Domain" type = "xsd:string" minOccurs = "0"/>
        </xsd:all>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "Credentials">
    <xsd:complexType>
        <xsd:all>
            <xsd:element name = "Holder"/>
            <xsd:choice>
                <xsd:element ref = "Login"/>
                <xsd:element ref = "dsig:X509Data"/>
                <xsd:element ref = "dsig:KeyValue"/>
                <xsd:any namespace = "##other" processContents = "strict"/>
                <xsd:element name = "NoCredentials"/>
            </xsd:choice>
        </xsd:all>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "AuthData">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref = "AuthType"/>
            <xsd:choice>
                <xsd:element ref = "UserHandle"/>
                <xsd:element ref = "IdentityToken"/>
```

```
                <xsd:element ref = "dsig:X509Data"/>
                <xsd:element ref = "dsig:KeyValue"/>
                <xsd:any namespace = "##other" processContents = "strict"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "AuthType">
    <xsd:complexType>
        <xsd:choice>
            <xsd:any namespace = "##other" processContents = "strict"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "UserHandle">
    <xsd:complexType>
        <xsd:all>
            <xsd:element name = "Directory" type = "xsd:string"/>
            <xsd:element name = "X509.DN" type = "xsd:string"/>
        </xsd:all>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "IdentityToken" type = "xsd:string"/>
<xsd:element name = "Entitlement">
    <xsd:complexType>
        <xsd:all>
            <xsd:element ref = "ID"/>
            <xsd:element ref = "Issuer"/>
            <xsd:element ref = "Date"/>
            <xsd:element ref = "Audiences" minOccurs = "0"/>
            <xsd:element ref = "DependsOn"/>
            <xsd:element ref = "AzData"/>
            <xsd:element name = "ValidityInterval" type = "ValidityInterval" minOccurs =
"0"/>
            <xsd:element ref = "dsig:Signature"/>
        </xsd:all>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "AzRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref = "ID"/>
            <xsd:element ref = "Time"/>
            <xsd:element ref = "NameAssertion"/>
            <xsd:element ref = "Question"/>
            <xsd:element ref = "Entitlement" minOccurs = "0" maxOccurs = "unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "AzResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref = "ID"/>
            <xsd:element ref = "Time"/>
            <xsd:element ref = "InResponseTo"/>
            <xsd:element ref = "Answer"/>
            <xsd:element ref = "entitlement" minOccurs = "0" maxOccurs = "unbounded"/>
```

```
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "ResourceContext">
        <xsd:complexType>
            <xsd:all>
                <xsd:element name = "Resource" type = "xsd:uriReference"/>
                <xsd:element name = "Method" type = "xsd:uriReference"/>
            </xsd:all>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "AzData">
        <xsd:complexType>
            <xsd:all>
                <xsd:any namespace = "##other" processContents = "strict"/>
            </xsd:all>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "Question">
        <xsd:complexType>
            <xsd:choice>
                <xsd:element ref = "ResourceContext"/>
                <xsd:any namespace = "##other" processContents = "strict"/>
            </xsd:choice>
            <xsd:attribute name = "AzModel" type = "xsd:uriReference"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "Answer">
        <xsd:complexType>
            <xsd:all>
                <xsd:element name = "Result" type = "ResultCode"/>
                <xsd:any namespace = "##other" processContents = "strict"/>
            </xsd:all>
            <xsd:attribute name = "AzModel" type = "xsd:uriReference"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Appendix C References

[RFC2068] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, *Hypertext Transfer Protocol -- HTTP/1.1*, RFC2068, January 1997, http://www.ietf.org/rfc/rfc2068.txt

[RFC2119] S. Bradner, editor. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. http://www.ietf.org/rfc/rfc2119.txt

[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] M. Wahl, S. Kille, T. Howes, Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names, RFC2253, December 1997, http://www.rfc-editor.org/rfc/rfc2253.txt

[X-TASS] Phillip Hallan-Baker, X-TASS: XML Trust Assertion Service Specification, Version 0.9, January 2001.

[MIME] Freed, N., and N. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, RFC 2045, Innosoft, First Virtual, November 1996, http://www.rfc-editor.org/rfc/rfc2045.txt

Appendix D Legal Notices

Copyright

Intellectual Property Statement

Neither the authors of this document nor their companies take any position regarding the validity or scope of any intellectual property or other rights that might be claimed pertaining to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither do they represent that they have made any effort to identify any such rights. Use of and results from the documents, specifications, technology and/or information contained herein is solely at the discretion and risk of the user.

Disclaimer

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