Digital Signature Service Core Protocols, Elements, and Bindings

2nd Committee Draft, 24 December 2004 (Working Draft 30)

Document identifier:
oasis-dss-1.0-core-spec-cd-02

Location:
http://docs.oasis-open.org/dss/

Editor:
Trevor Perrin, *individual* <trevp@trevp.net>

Contributors:
Dimitri Andivahis, Surety
Juan Carlos Cruellas, *individual*
Frederick Hirsch, Nokia
Pieter Kasselman, Cybertrust
Andreas Kuehne, *individual*
Paul Madsen, Entrust
John Messing, American Bar Association
Tim Moses, Entrust
Nick Pope, *individual*
Rich Salz, DataPower
Ed Shallow, Universal Postal Union

Abstract:
This document defines XML request/response protocols for signing and verifying XML documents and other data. It also defines an XML timestamp format, and an XML signature property for use with these protocols. Finally, it defines transport and security bindings for the protocols.

Status:
This is a **Committee Draft** produced by the OASIS Digital Signature Service Technical Committee. Committee members should send comments on this draft to dss@lists.oasis-open.org.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Digital Signature Service TC web page at http://www.oasis-open.org/committees/dss/ipr.php.

Table of Contents

1 Introduction

This specification defines the XML syntax and semantics for the Digital Signature Service core protocols, and for some associated core elements. The core protocols support the server-based creation and verification of different types of signatures and timestamps. The core elements include an XML timestamp format, and an XML signature property to contain a representation of a client's identity.

The core protocols are typically *bound* into other protocols for transport and security, such as HTTP and TLS. This document provides an initial set of bindings. The core protocols are also typically *profiled* to constrain optional features and add additional features. Other documents will provide profiles of DSS.

The following sections describe how to understand the rest of this specification.

1.1 Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 **[RFC 2119]**. These keywords are capitalized when used to unambiguously specify requirements over protocol features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

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

```
Listings of DSS schemas appear like this.
```

1.2 Schema Organization and Namespaces

The structures described in this specification are contained in the schema file **[Core-XSD]**. All schema listings in the current document are excerpts from the schema file. In the case of a disagreement between the schema file and this document, the schema file takes precedence.

This schema is associated with the following XML namespace:

```
http://docs.oasis-open.org/dss/oasis-dss-1.0-core-schema-cd-02.xsd
```

If a future version of this specification is needed, it will use a different namespace.

Conventional XML namespace prefixes are used in the schema:

- The prefix `dss:` stands for the DSS core namespace **[Core-XSD]**.
- The prefix `ds:` stands for the W3C XML Signature namespace **[XMLSig]**.
- The prefix `xs:` stands for the W3C XML Schema namespace **[Schema1]**.
- The prefix `saml:` stands for the OASIS SAML Schema namespace **[SAMLCore1.1]**.

Applications MAY use different namespace prefixes, and MAY use whatever namespace defaulting/scoping conventions they desire, as long as they are compliant with the Namespaces in XML specification **[XML-ns]**.

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

```
<xs:schema targetNamespace="http://www.docs.oasis-open.org/dss/oasis-
dss-1.0-core-schema-cd-2.xsd"
    xmlns:dss="http://www.docs.oasis-open.org/dss/oasis-dss-1.0-core-
schema-cd-2.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
```

1.3 DSS Overview (Non-normative)

This specification describes two XML-based request/response protocols – a signing protocol and a verifying protocol. Through these protocols a client can send documents to a server and receive back a signature on the documents; or send documents and a signature to a server, and receive back an answer on whether the signature verifies the documents.

These operations could be useful in a variety of contexts – for example, they could allow clients to access a single corporate key for signing press releases, with centralized access control, auditing, and archiving of signature requests. They could also allow clients to create and verify signatures without needing complex client software and configuration.

The signing and verifying protocols are chiefly designed to support the creation and verification of XML signatures **[XMLSig]**, XML timestamps (see section 5.1), and CMS signatures **[RFC3369]**. These protocols may also be extensible to other types of signatures and timestamps, such as PGP signatures **[RFC 2440]** or RFC 3161 TimeStampTokens.

It is expected that the signing and verifying protocols will be *profiled* to meet many different application scenarios. In anticipation of this, these protocols have only a minimal set of required elements, which deal with transferring “input documents” and signatures back and forth between client and server. The input documents to be signed or verified can be transferred in their entirety, or the client can hash the documents itself and only send the hash values, to save bandwidth and protect the confidentiality of the document content.

All functionality besides transferring input documents and signatures is relegated to a framework of “optional inputs” and “optional outputs”. This document defines a number of optional inputs and outputs. Profiles of these protocols can pick and choose which optional inputs and outputs to support, and can introduce their own optional inputs and outputs when they need functionality not anticipated by this specification.

Examples of optional inputs to the signing protocol include: what type of signature to produce, which key to sign with, who the signature is intended for, and what signed and unsigned properties to place in the signature. Examples of optional inputs to the verifying protocol include: the time for which the client would like to know the signature’s validity status, additional validation data necessary to verify the signature (such as certificates and CRLs), and requests for the server to return information such as the signer’s name or the signing time.

The signing and verifying protocol messages must be transferred over some underlying protocol(s) which provide message transport and security. A *binding* specifies how to use the signing and verifying protocols with some underlying protocol, such as HTTP POST or TLS. Section 6 provides an initial set of bindings.

In addition to defining the signing and verifying protocols, this specification defines two XML elements that are related to these protocols. First, an XML timestamp element is defined in section 5.1. The signing and verifying protocols can be used to create and verify XML timestamps; a profile for doing so is defined in **[XML-TSP]**. Second, a Requester Identity element is defined in section 5.2. This element can be used as a signature property in an XML signature, to give the name of the end-user who requested the signature.

2 Common Protocol Structures

The following sections describe XML structures and types that are used in multiple places.

2.1 Type AnyType

The **AnyType** complex type allows arbitrary XML content within an element.

```
<xs:complexType name="AnyType">
    <xs:sequence>
        <xs:any processContents="lax" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

2.2 Type InternationalStringType

The **InternationalStringType** complex type attaches an xml:lang attribute to a human-readable string to specify the string's language.

```
<xs:complexType name="InternationalStringType">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute ref="xml:lang" use="required">
        </xs:extension base="xs:string">
    </xs:simpleContent>
</xs:complexType>
```

2.3 Type saml:NameIdentifierType

The **saml:NameIdentifierType** complex type is used where different types of names are needed (such as email addresses, Distinguished Names, etc.). This type is borrowed from **[SAMLCore1.1]** section 2.4.2.2. It consists of a string with the following attributes:

`NameQualifier` [Optional]

The security or administrative domain that qualifies the name of the subject. This attribute provides a means to federate names from disparate user stores without collision.

`Format` [Optional]

A URI reference representing the format in which the string is provided. See section 7.3 of **[SAMLCore1.1]** for some URI references that may be used as the value of the `Format` attribute.

2.4 Element <InputDocuments>

The `<InputDocuments>` element is used to send input documents to a DSS server, whether for signing or verifying. An input document can be any piece of data that can be used as input to a signature or timestamp calculation. An input document can even *be* a signature or timestamp (for example, a pre-existing signature can be counter-signed or timestamped). An input document could also be a `<ds:Manifest>`, allowing the client to handle manifest creation and verification while using the server to create and verify the rest of the signature.

The `<InputDocuments>` element consists of any number of the following elements:

`<Document>` [Any Number]

An XML document or some other data.

`<DocumentHash>` [Any Number]

A hash value of an XML document or some other data.

```
<xs:element name="InputDocuments">
    <xs:complexType>
        <xs:sequence>
            <xs:choice minOccurs="1" maxOccurs="unbounded">
                <xs:element ref="dss:Document"/>
                <xs:element ref="dss:DocumentHash"/>
                <xs:any processContents="lax"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

When using DSS to create or verify XML signatures, each input document will usually correspond to a single `<ds:Reference>` element. Thus, in our descriptions below of the `<Document>` and `<DocumentHash>` elements, we will explain how certain elements and attributes of a `<Document>` or `<DocumentHash>` correspond to components of a `<ds:Reference>`.

2.4.1 Type DocumentBaseType

The **DocumentBaseType** complex type is subclassed by both the `<Document>` and `<DocumentHash>` elements. It contains the following elements and attributes:

`ID` [Optional]

This identifier gives the input document a unique label within a particular request message. Through this identifier, an optional input (see section 2.5) can refer to a particular input document.

`RefURI` [Optional]

This specifies the value for a `<ds:Reference>` element's `URI` attribute when referring to this input document. The `RefURI` attribute SHOULD be specified; no more than one `RefURI` attribute may be omitted in a single signing request.

`RefType` [Optional]

This specifies the value for a `<ds:Reference>` element's `Type` attribute when referring to this input document.

`<ds:Transforms>` [Optional]

This specifies the value for a `<ds:Reference>` element's `<ds:Transforms>` child element when referring to this input document. In other words, this specifies transforms that the client has already applied to the input document. In the case of a `<Document>` (but not a `<DocumentHash>`) the server may apply additional transforms, which will be appended to the ones specified here to produce the final `<ds:Transforms>` list.

`<Schema>` [Optional]

This may be used when the document contains XML signatures. It transfers an XML Schema **[Schema1]** which gives the ID attributes of elements within the input document, which may be necessary if the included signatures' `<ds:Reference>` elements use XPointer expressions. See section 4.3, step 2 for details.

```
<xs:complexType name="DocumentBaseType" abstract="true">
    <xs:sequence>
        <xs:element ref="ds:Transforms" minOccurs="0"/>
        <xs:element name="Schema" type="xs:base64Binary"
                    minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="ID" type="xs:ID" use="optional"/>
    <xs:attribute name="RefURI" type="xs:ID" use="optional"/>
    <xs:attribute name="RefType" type="xs:ID" use="optional"/>
</xs:complexType>
```

2.4.2 Element <Document>

The `<Document>` element may contain the following elements (in addition to the common ones listed in section 2.4.1):

`<XMLData>` [Optional]

This contains arbitrary XML content.

`<Base64Data>` [Optional]

This contains a base64 encoding of an XML document or some other data. The type of data is specified by its `MimeType` attribute. The `MimeType` attribute is not required for XML signatures, but may be required when using DSS with other signature types.

The document hash for signing is created from the element content of `<XMLData>` (i.e. the `<XMLData>` tags are not included), or from the content of the `<Base64Data>` element after it is base64 decoded.

```
<xs:element name="Document">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="dss:DocumentBaseType">
                <xs:choice>
                    <xs:element ref="dss:XMLData"/>
                    <xs:element ref="dss:Base64Data"/>
                </xs:choice>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

<xs:element name="XMLData" type="dss:AnyType"/>

<xs:element name="Base64Data">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:base64Binary">
                <xs:attribute name="MimeType" type="xs:string"
                              use="optional">
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
```

2.4.3 Element <DocumentHash>

The `<DocumentHash>` element contains the following elements (in addition to the common ones listed in section 2.4.1):

`<ds:DigestMethod>` [Required]

This identifies the digest algorithm used to hash the document. This specifies the value for a `<ds:Reference>` element's `<ds:DigestMethod>` child element when referring to this input document.

`<ds:DigestValue>` [Required]

This gives the document's hash value. This specifies the value for a `<ds:Reference>` element's `<ds:DigestValue>` child element when referring to this input document.

```
<xs:element name="DocumentHash">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="dss:DocumentBaseType">
                <xs:sequence>
                    <xs:element ref="ds:DigestMethod"/>
                    <xs:element ref="ds:DigestValue"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
```

2.5 Element <SignatureObject>

The `<SignatureObject>` element contains a signature or timestamp of some sort. This element is returned in a sign response message, and sent in a verify request message. It may contain one of the following child elements:

`<ds:Signature>` [Optional]

An XML signature **[XMLSig]**.

`<Timestamp>` [Optional]

An XML timestamp (see section 5.1).

`<Base64Signature>` [Optional]

A base64 encoding of some non-XML signature, such as a PGP **[RFC 2440]** or CMS **[RFC 3369]** signature. The type of signature is specified by its `Type` attribute (see section 7.1).

`<SignaturePtr>` [Optional]

This is used in a verify request to point to an XML signature in an input document.

`Schema` [Optional]

This may be used in conjunction with an enveloping XML signature to transfer an XML Schema **[Schema1]** which gives the ID attributes of elements enveloped within the signature. This may be necessary to allow the signature's `<ds:Reference>` elements which refer to these enveloped elements with XPointer expressions to be resolved. See section 4.3, step 2 for details.

A `<SignaturePtr>` contains the following attributes:

`WhichDocument` [Required]

This identifies the input document being pointed at.

`XPath` [Optional]

This identifies the element being pointed at. The XPath expression is evaluated from the context node identified by the `WhichDocument` attribute.

The following schema fragment defines the `<SignatureObject>`, `<Base64Signature>`, and `<SignaturePtr>` elements:

```
<xs:element name="SignatureObject">
    <xs:complexType>
        <xs:sequence>
            <xs:choice>
                <xs:element ref="ds:Signature"/>
                <xs:element ref="dss:Timestamp"/>
                <xs:element ref="dss:Base64Signature"/>
                <xs:element ref="dss:SignaturePtr"/>
                <xs:any processContents="lax"/>
            </xs:choice>
            <xs:element name="Schema" type="xs:base64Binary"
                    minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="Base64Signature">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:base64Binary">
                <xs:attribute name="Type" type="xs:anyURI"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="SignaturePtr">
    <xs:complexType>
        <xs:attribute name="WhichDocument" type="xs:IDREF"/>
        <xs:attribute name="XPath" type="xs:string" use="optional"/>
     </xs:complexType>
</xs:element>
```

2.6 Element <Result>

The `<Result>` element is returned with every response message. It contains the following child elements:

`<ResultMajor>` [Required]

The most significant component of the result code.

`<ResultMinor>` [Optional]

The least significant component of the result code.

`<ResultMessage>` [Optional]

A message which MAY be returned to an operator, logged, used for debugging, etc.

```
<xs:element name="Result">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ResultMajor" type="xs:anyURI"/>
            <xs:element name="ResultMinor" type="xs:anyURI"
                        minOccurs="0"/>
            <xs:element name="ResultMessage"
                        type="InternationalStringType" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

The `<ResultMajor>` and `<ResultMinor>` URIs MUST be values defined by this specification or by some profile of this specification. The `<ResultMajor>` values defined by this specification are:

`urn:oasis:names:tc:dss:1.0:resultmajor:Success`

The protocol executed succesfully.

`urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError`

The request could not be satisfied due to an error on the part of the requester.

`urn:oasis:names:tc:dss:1.0:resultmajor:ResponderError`

The request could not be satisfied due to an error on the part of the responder.

This specification defines the following two `<ResultMinor>` values. These values SHALL only be returned when the `<ResultMajor>` code is `RequesterError`:

`urn:oasis:names:tc:dss:1.0:resultminor:NotAuthorized`

The client is not authorized to perform the request.

`urn:oasis:names:tc:dss:1.0:resultminor:NotSupported`

The server didn't recognize or doesn't support some aspect of the request.

The `Success` `<ResultMajor>` code on a verify response message SHALL be followed by a `<ResultMinor>` code which indicates the status of the signature. See section 4 for details.

2.7 Elements <OptionalInputs> and <OptionalOutputs>

All request messages can contain an `<OptionalInputs>` element, and all response messages can contain an `<OptionalOutputs>` element. Several optional inputs and outputs are defined in this document, and profiles can define additional ones.

The `<OptionalInputs>` contains additional inputs associated with the processing of the request. Profiles will specify which optional inputs are allowed and what their default values are. All optional inputs in a profile SHOULD have some default value, so that a client may omit the `<OptionalInputs>` yet still get service from any profile-compliant DSS server. If a server doesn't recognize or can't handle any optional input, it MUST reject the request with a `<ResultMajor>` code of `RequesterError` and a `<ResultMinor>` code of `NotSupported` (see section 2.6).

The `<OptionalOutputs>` element contains additional protocol outputs. The client MAY request the server to respond with certain optional outputs by sending certain optional inputs. The server MAY also respond with outputs the client didn't request, depending on the server's profile and policy.

The `<OptionalInputs>` and `<OptionalOutputs>` elements contain unordered inputs and outputs. Applications MUST be able to handle optional inputs or outputs appearing in any order within these elements. Normally, there will only be at most one occurrence of any particular optional input or output within a protocol message. Where multiple occurrences of an optional input or optional output are allowed, it will be explicitly specified (see section 4.6.8 for an example).

The following schema fragment defines the `<OptionalInputs>` and `<OptionalOutputs>` elements:

```
<xs:element name="OptionalInputs" type="dss:AnyType"/>

<xs:element name="OptionalOutputs" type="dss:AnyType"/>
```

2.8 Common Optional Inputs

These optional inputs can be used with both the signing protocol and the verifying protocol.

2.8.1 Optional Input <ServicePolicy>

The `<ServicePolicy>` element indicates a particular policy associated with the DSS service. The policy may include information on the characteristics of the server that are not covered by the `Profile` attribute (see sections 3.1 and 4.1). The `<ServicePolicy>` element may be used to select a specific policy if a service supports multiple policies for a specific profile, or as a sanity-check to make sure the server implements the policy the client expects.

```
<xs:element name="ServicePolicy" type="xs:anyURI"/>
```

2.8.2 Optional Input <ClaimedIdentity>

The `<ClaimedIdentity>` element indicates the identity of the client who is making a request. The server may use this to parameterize any aspect of its processing. Profiles that make use of this element MUST define its semantics.

The `<SupportingInfo>` child element can be used by profiles to carry information related to the claimed identity. One possible use of `<SupportingInfo>` is to carry authentication data that authenticates the request as originating from the claimed identity (examples of authentication data include a password or SAML Assertion **[SAMLCore1.1]**, or a signature or MAC calculated over the request using a client key).

The claimed identity may be authenticated using the security binding, according to section 6, or using authentication data provided in the `<SupportingInfo>` element. The server MUST check that the asserted `<Name>` is authenticated before relying upon the `<Name>`.

```
<xs:element name="ClaimedIdentity">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Name" type="saml:NameIdentifierType"/>
            <xs:element name="SupportingInfo" type="dss:AnyType"
                        minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

2.8.3 Optional Input <Language>

The `<Language>` element indicates which language the client would like to receive **InternationalStringType** values in. The server should return appropriately localized strings, if possible.

```
<xs:element name="Language" type="xs:language"/>
```

2.8.4 Optional Input <AdditionalProfile>

The `<AdditionalProfile>` element can appear multiple times in a request. It indicates additional profiles which modify the main profile specified by the `Profile` attribute (thus the `Profile` attribute MUST be present; see sections 3.1 and 4.1 for details of this attribute). The interpretation of additional profiles is determined by the main profile.

```
<xs:element name="AdditionalProfile" type="xs:anyURI"/>
```

3 The DSS Signing Protocol

3.1 Element <SignRequest>

The `<SignRequest>` element is sent by the client to request a signature or timestamp on some input documents. It contains the following attributes and elements:

`RequestID` [Optional]

This attribute is used to correlate requests with responses. When present in a request, the server MUST return it in the response.

`Profile` [Optional]

This attribute indicates a particular DSS profile. It may be used to select a profile if a server supports multiple profiles, or as a sanity-check to make sure the server implements the profile the client expects.

`<OptionalInputs>` [Optional]

Any additional inputs to the request.

`<InputDocuments>` [Required]

The input documents which the signature will be calculated over.

```
<xs:element name="SignRequest">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="dss:OptionalInputs" minOccurs="0"/>
            <xs:element ref="dss:InputDocuments"/>
        </xs:sequence>
        <xs:attribute name="RequestID" type="xs:string"
                      use="optional"/>
        <xs:attribute name="Profile" type="xs:anyURI" use="optional"/>
    </xs:complexType>
</xs:element>
```

3.2 Element <SignResponse>

The `<SignResponse>` element contains the following attributes and elements:

`RequestID` [Optional]

This attribute is used to correlate requests with responses. When present in a request, the server MUST return it in the response.

`Profile` [Optional]

This attribute indicates the particular DSS profile used by the server. It may be used by the client for logging purposes or to make sure the server implements a profile the client expects.

`<Result>` [Required]

A code representing the status of the request.

`<OptionalOutputs>` [Optional]

Any additional outputs returned by the server.

`<SignatureObject>` [Optional]

The resultant signature or timestamp, if the request succeeds. This MUST NOT contain a `<SignaturePtr>`; it MUST contain an entire signature or timestamp.

```
<xs:element name="SignResponse">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="dss:Result"/>
            <xs:element ref="dss:OptionalOutputs" minOccurs="0"/>
            <xs:element ref="dss:SignatureObject" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="RequestID" type="xs:string"
                      use="optional"/>
        <xs:attribute name="Profile" type="xs:anyURI" use="required"/>
    </xs:complexType>
</xs:element>
```

3.3 Basic Processing for XML Signatures

A DSS server that produces XML signatures SHOULD perform the following steps, upon receiving a `<SignRequest>`. These steps may be changed or overridden by the optional inputs (for example, see section 3.5.5), or by the profile or policy the server is operating under.

1. The server hashes the contents of each `<Document>`, as follows:
    a. If the `<Document>` contains `<XMLData>`, the server converts the XML content of the `<XMLData>` element into an octet stream using Canonical XML **[XML-C14N]**.
    b. If the `<Document>` contains `<Base64Data>`, the server base64-decodes the text content of the `<Base64Data>` into an octet stream.
    c. If the server wishes, it may apply additional XML signature transforms to the octet stream produced in a or b. These transforms should be applied as per **[XMLSig]** section 4.3.3.2. Following **[XMLSig]**, if the end result of these transforms is an XML node set, the server must convert the node set back into an octet stream using Canonical XML **[XML-C14N]**.
    d. The server hashes the resultant octet stream.
2. The server forms a `<ds:Reference>` for each input document. The elements and attributes of the `<ds:Reference>` are set as follows:
    a. The `URI` attribute is set to the input document's `RefURI` attribute. If the input document has no `RefURI` attribute, the `<ds:Reference>` element's `URI` attribute is omitted. A signature MUST NOT be created if more than one `RefURI` is omitted in the set of input documents.
    b. The `Type` attribute is set to the input document's `RefType` attribute. If the input document has no `RefType` attribute, the `<ds:Reference>` element's `Type` attribute is omitted.
    c. The `<ds:DigestMethod>` element is set to the hash method that was used in step 1.d (for a `<Document>`), or to the input document's `<ds:DigestMethod>` (for a `<DocumentHash>`).
    d. The `<ds:DigestValue>` element is set to the hash value that was calculated in step 1.d (for a `<Document>`), or to the input document's `<ds:DigestValue>` (for a `<DocumentHash>`).

    e. The `<ds:Transforms>` element is set to the input document's `<ds:Transforms>` element. If any additional transforms were applied by the server in step 1.c., these are appended as well.

3. The server creates an XML signature using the `<ds:Reference>` elements created in Step 2, according to the processing rules in **[XMLSig]**.

3.3.1 Enveloping Signatures

A client can use any server that implements basic processing, as defined above, to create an enveloping XML signature. To do this, the client simply splices the to-be-enveloped document(s) into the returned `<ds:Signature>`.

3.3.2 Enveloped Signatures

A client can use any server that implements basic processing, as defined above, to create an enveloped XML signature. To do this, the client simply indicates an Enveloped Signature Transform **[XMLSig]** on the appropriate input document, and splices the returned `<ds:Signature>` into the appropriate document.

A client who desires an enveloped signature can also use the `<SignaturePlacement>` optional input to instruct the server to insert the resultant signature into one of the input documents, and return the resultant document as an optional output. See section 3.5.7 for details.

3.4 Basic Processing for CMS Signatures

A DSS server that produces CMS signatures **[RFC 3369]** SHOULD perform the following steps, upon receiving a `<SignRequest>`. These steps may be changed or overridden by the optional inputs, or by the profile or policy the server is operating under.

The `<SignRequest>` should contain either a single `<Document>` or a single `<DocumentHash>`:

1. If a `<Document>` is present, the server hashes its contents as follows:
    a. If the `<Document>` contains `<XMLData>`, the server extracts the text content of the `<XMLData>` as an octet stream.
    b. If the `<Document>` contains `<Base64Data>`, the server base64-decodes the text content of the `<Base64Data>` into an octet stream.
    c. The server hashes the resultant octet stream.
2. The server forms a `SignerInfo` structure based on the input document. The components of the `SignerInfo` are set as follows:
    a. The `digestAlgorithm` field is set to the OID value for the hash method that was used in step 1.c (for a `<Document>`), or to the OID value that is equivalent to the input document's `<ds:DigestMethod>` (for a `<DocumentHash>`).
    b. The `signedAttributes` field's `message-digest` attribute contains the hash value that was calculated in step 1.c (for a `<Document>`), or that was sent in the input document's `<ds:DigestValue>` (for a `<DocumentHash>`). Other `signedAttributes` may be added by the server, according to its profile or policy, or according to the `<Properties>` optional input (see section 3.5.6).
    c. The remaining fields (`sid`, `signatureAlgorithm`, and `signature`) are filled in as per a normal CMS signature.

3. The server creates a CMS signature (i.e. a `SignedData` structure) containing the `SignerInfo` that was created in Step 2. The resulting `SignedData` should be detached (i.e. external) unless the client sends the `<EnvelopingSignature>` optional input (see section 3.5.8).

3.5 Optional Inputs and Outputs

This section defines some optional inputs and outputs that profiles of the DSS signing protocol might find useful. Section 2.8 defines some common optional inputs that can also be used with the signing protocol. Profiles of the signing protocol can define their own optional inputs and outputs, as well. General handling of optional inputs and outputs is discussed in section 2.7.

3.5.1 Optional Input <SignatureType>

The `<SignatureType>` element indicates the type of signature or timestamp to produce (such as an XML signature, an XML timestamp, a CMS signature, etc.). See section 7.1 for some URI references that MAY be used as the value of this element.

```
<xs:element name="SignatureType" type="xs:anyURI"/>
```

3.5.2 Optional Input <AddTimestamp>

The `<AddTimestamp>` element indicates that the client wishes the server to provide a timestamp as a property or attribute of the resultant signature. The `Type` attribute, if present, indicates what type of timestamp to apply. Profiles that use this optional input MUST define the allowed values, and the default value, for the `Type` attribute (unless only a single type of timestamp is supported, in which case the `Type` attribute can be omitted).

```
<xs:element name="AddTimestamp">
    <xs:complexType>
        <xs:attribute name="Type" type="xs:anyURI" use="optional"/>
    </xs:complexType>
</xs:element>
```

3.5.3 Optional Input <IntendedAudience>

The `<IntendedAudience>` element tells the server who the target audience of this signature is is. The server may use this to parameterize any aspect of its processing (for example, the server may choose to sign with a key that it knows a particular recipient trusts).

```
<xs:element name="IntendedAudience">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Recipient" type="saml:NameIdentifierType"
                        maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

3.5.4 Optional Input <KeySelector>

The `<KeySelector>` element tells the server which key to use.

```
<xs:element name="KeySelector">
    <xs:complexType>
        <xs:choice>
            <xs:element ref="ds:KeyInfo"/>
            <xs:any processContents="lax"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

3.5.5 Optional Input <SignedReferences>

The `<SignedReferences>` element gives the client greater control over how the `<ds:Reference>` elements are formed. When this element is present, steps 1 and 2 of Basic Processing (section 3.3) are overridden. Instead of there being a one-to-one correspondence between input documents and `<ds:Reference>` elements, now each `<SignedReference>` element controls the creation of a corresponding `<ds:Reference>`.

Since each `<SignedReference>` refers to an input document, this allows multiple `<ds:Reference>` elements to be based on a single input document. Furthermore, the client can request additional transforms to be applied to each `<ds:Reference>`, and can set each `<ds:Reference>` element's `Id` attribute. These aspects of the `<ds:Reference>` can only be set through the `<SignedReferences>` optional input; they cannot be set through the input documents, since they are aspects of the reference to the input document, not the input document itself.

Each `<SignedReference>` element contains:

`WhichDocument` [Required]

Which input document this reference refers to (see the `ID` attribute in section 2.4.1).

`RefId` [Optional]

Sets the `Id` attribute on the corresponding `<ds:Reference>`.

`<ds:Transforms>` [Optional]

Requests the server to perform additional transforms on this reference.

When the `<SignedReferences>` optional input is present, steps 1 and 2 of Basic Processing are replaced with the steps below:

1. The server prepares a hash value for each `<SignedReference>`, as follows:
    a. If the referenced input document is a `<DocumentHash>`, the server is done.
    b. Otherwise, if the referenced `<Document>` contains `<XMLData>`, the server extracts the text content of the `<XMLData>` as an octet stream.
    c. If the referenced `<Document>` contains `<Base64Data>`, the server base64-decodes the text content of the `<Base64Data>` into an octet stream.
    d. The server applies the XML signature transforms indicated by the `<SignedReference>` to the octet stream produced in b or c. These transforms should be applied as per **[XMLSig]** section 4.3.3.2.
    e. If the server wishes, it may apply additional XML signature transforms to the octet stream produced in d.

        f. If the end result of these transforms is an XML node set, the server must convert the node set back into an octet stream using Canonical XML **[XML-C14N]**.

        g. The server hashes the resultant octet stream.

2. The server forms a `<ds:Reference>` for each `<SignedReference>`. The elements and attributes of the `<ds:Reference>` are set as follows:

    a. The `URI` attribute is set to the referenced input document's `RefURI` attribute. If the input document has no `RefURI` attribute, the `<ds:Reference>` element's `URI` attribute is omitted. A signature MUST NOT be created if more than one `RefURI` is omitted in the set of input documents, or if more than one `<SignedReference>` refers to an input document that has no `RefURI`.

    b. The `Type` attribute is set to the referenced input document's `RefType` attribute. If the input document has no `RefType` attribute, the `<ds:Reference>` element's `Type` attribute is omitted.

    c. The `Id` attribute is set to the `<SignedReference>` element's `RefId` attribute. If the `<SignedReference>` has no `RefId` attribute, the `<ds:Reference>` element's `Id` attribute is omitted.

    d. The `<ds:DigestMethod>` element is set to the hash method that was used in step 1.g (for a `<Document>`), or to the referenced input document's `<ds:DigestMethod>` (for a `<DocumentHash>`).

    e. The `<ds:DigestValue>` element is set to the hash value that was calculated in step 1.g (for a `<Document>`), or to the referenced input document's `<ds:DigestValue>` (for a `<DocumentHash>`).

    f. The `<ds:Transforms>` element is set to the referenced input document's `<ds:Transforms>` element with the `<SignedReference>` element's transforms appended. If any additional transforms were applied by the server in step 1.e., these are appended as well.

```
<xs:element name="SignedReferences">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="dss:SignedReference"
                        maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="SignedReference">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="ds:Transforms" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="WhichDocument" type="xs:IDREF"
                      use="required"/>
        <xs:attribute name="RefId" type="xs:string" use="optional"/>
    </xs:complexType>
</xs:element>
```

3.5.6 Optional Input <Properties>

The `<Properties>` element is used to request that the server add certain signed or unsigned properties (aka "signature attributes") into the signature. The client can send the server a particular value to use for each property, or leave the value up to the server to determine. The server can add additional properties, even if these aren't requested by the client.

The `<Properties>` element contains:

`<SignedProperties>` [Optional]

These properties will be covered by the signature.

`<UnsignedProperties>` [Optional]

These properties will not be covered by the signature.

Each `<Property>` element contains:

`<Identifier>` [Required]

A URI reference identifying the property.

`<Value>` [Optional]

If present, the value the server should use for the property.

This specification does not define any properties. Profiles that make use of this element MUST define the allowed property URIs and their allowed values.

```
<xs:element name="Properties">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="SignedProperties"
                        type="dss:PropertiesType" minOccurs="0"/>
            <xs:element name="UnsignedProperties"
                        type="dss: PropertiesType" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:complexType name="PropertiesType">
    <xs:sequence>
        <xs:element ref="dss:Property" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:element name="Property">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Identifier" type="xs:anyURI"/>
            <xs:element name="Value" type="dss:AnyType"
                        minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

3.5.7 Optional Input <SignaturePlacement> and Output <DocumentWithSignature>

The `<SignaturePlacement>` element instructs the server to place the signature inside one of the input documents, and return the resulting document. In the case of an XML input document, the client may instruct the server precisely where to place the signature with the optional `<XpathAfter>` and `<XpathFirstChildOf>` child elements. In the case of a non-XML input document, or when these child elements are omitted, then the server will decide how to place the signature in the input document. The `<SignaturePlacement>` element contains the following attributes and elements:

`WhichDocument` [Required]

Identifies the input document which the signature will be inserted into (see the `ID` attribute in section 2.4.1).

`<XpathAfter>` [Optional]

Identifies an element, in the XML input document, which the signature will be inserted after. The XPath expression is evaluated from the context node identified by the `WhichDocument` attribute. The signature is placed immediately after the end tag of the specified element.

`<XpathFirstChildOf>` [Optional]

Identifies an element, in the XML input document, which the signature will be inserted as the first child of. The XPath expression is evaluated from the context node identified by the `WhichDocument` attribute. The signature is placed immediately after the start tag of the specified element.

```
<xs:element name=”SignaturePlacement”>
    <xs:complexType>
        <xs:choice>
            <xs:element name=”XpathAfter” type=”xs:string”/>
            <xs:element name=”XpathFirstChildOf” type=”xs:string”/>
        </xs:choice>
        <xs:attribute name=”WhichDocument” type=”xs:IDREF”/>
    </xs:complexType>
</xs:element>
```

The `<DocumentWithSignature>` optional output contains the input document with the signature inserted. It has one child element:

`<Document>` [Optional]

This contains the input document with a signature inserted in some fashion.

```
<xs:element name=”DocumentWithSignature”>
    <xs:complexType>
        <xs:sequence>
            <xs:element ref=”Document”/>
        <xs:sequence>
    </xs:complexType>
</xs:element>
```

3.5.8 Optional Input <EnvelopingSignature>

The `<EnvelopingSignature>` element causes the server to incorporate one of the input documents inside the returned signature. In the case of an XML signature, the input document MUST be XML and will be placed inside a `<ds:Object>` element. In the case of a CMS signature, the input document’s decoded octet stream will be included as encapsulated content.

`WhichDocument` [Required]

Identifies the XML input document which will be inserted into the returned signature (see the `ID` attribute in section 2.4.1).

`ObjId` [Optional]

Sets the `Id` attribute on the returned `<ds:Object>`.

```
<xs:element name=”EnvelopingSignature”>
    <xs:complexType>
        <xs:attribute name=”WhichDocument” type=”xs:IDREF”/>
        <xs:attribute name=”ObjId” type=”xs:string” use=”optional”/>
    </xs:complexType>
</xs:element>
```

4 The DSS Verifying Protocol

4.1 Element <VerifyRequest>

The `<VerifyRequest>` element is sent by the client to verify a signature or timestamp on some input documents. It contains the following attributes and elements:

`RequestID` [Optional]

This attribute is used to correlate requests with responses. When present in a request, the server MUST return it in the response.

`Profile` [Optional]

This attribute indicates a particular DSS profile. It may be used to select a profile if a server supports multiple profiles, or as a sanity-check to make sure the server implements the profile the client expects.

`<OptionalInputs>` [Optional]

Any additional inputs to the request.

`<SignatureObject>` [Optional]

This element contains a signature or timestamp, or else contains a `<SignaturePtr>` that points to an XML signature in one of the input documents. If this element is omitted, there must be only a single `<InputDocument>` which the server will search to find the to-be-verified signature(s). A `<SignaturePtr>` or omitted `<SignatureObject>` MUST be used whenever the to-be-verified signature is an XML signature which uses an Enveloped Signature Transform; otherwise the server would have difficulty locating the signature and applying the Enveloped Signature Transform.

`<InputDocuments>` [Optional]

The input documents which the signature was calculated over. The signature to be verified may also be contained in one of these documents. This element may be omitted if an enveloping signature inside the `<SignatureObject>` contains the input document(s).

```
<xs:element name="VerifyRequest">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="dss:OptionalInputs" minOccurs="0"/>
            <xs:element ref="dss:SignatureObject" minOccurs="0"/>
            <xs:element ref="dss:InputDocuments" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="RequestID" type="xs:string"
                      use="optional"/>
        <xs:attribute name="Profile" type="xs:anyURI" use="optional"/>
    </xs:complexType>
</xs:element>
```

4.2 Element <VerifyResponse>

The `<VerifyResponse>` element contains the following attributes and elements:

`RequestID` [Optional]

This attribute is used to correlate requests with responses. When present in a request, the server MUST return it in the response.

`Profile` [Optional]

This attribute indicates the particular DSS profile used by the server. It may be used by the client for logging purposes or to make sure the server implements a profile the client expects.

`<Result>` [Required]

A code representing the status of the corresponding request.

`<OptionalOutputs>` [Optional]

Any additional outputs returned by the server.

```
<xs:element name="VerifyResponse">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="dss:Result"/>
            <xs:element ref="dss:OptionalOutputs" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="RequestID" type="xs:string"
                      use="optional"/>
        <xs:attribute name="Profile" type="xs:anyURI" use="required"/>
    </xs:complexType>
</xs:element>
```

4.3 Basic Processing for XML Signatures

A DSS server that verifies XML signatures SHOULD perform the following steps, upon receiving a `<VerifyRequest>`. These steps may be changed or overridden by the optional inputs, or by the profile or policy the server is operating under. For more details on multi-signature verification, see section 4.3.1.

1. The server retrieves one or more `<ds:Signature>` objects, as follows: If the `<SignatureObject>` is present, the server retrieves either the `<ds:Signature>` that is a child element of the `<SignatureObject>`, or those `<ds:Signature>` objects which are pointed to by the `<SignaturePtr>` in the `<SignatureObject>`.

   If the `<SignaturePtr>` points to an input document but not a specific element in that document, the pointed-to input document must be a `<Document>` element containing XML either in an `<XMLData>` or `<Base64Data>` element. The server will search and find every `<ds:Signature>` element in this input document, and verify each `<ds:Signature>` according to the steps below.

   If the `<SignatureObject>` is omitted, there MUST be only a single `<Document>` element. This case is handled *as if* a `<SignaturePtr>` pointing to the single `<Document>` was present: the server will search and find every `<ds:Signature>` element in this input document, and verify each `<ds:Signature>` according to the steps below.

   For each `<ds:Reference>` in the `<ds:Signature>`, the server finds the input document with matching `RefURI` and `RefType` values. If such an input document isn't present, and the `<ds:Reference>` uses a null URI *without* a barename XPointer, then the relevant input document is the input document the `<ds:Signature>` is contained within, or the

`<ds:Signature>` itself if it is contained within the `<SignatureObject>` element. If the `<ds:Reference>` uses a null URI *with* a barename XPointer, the XPointer should be evaluated against the input document the `<ds:Signature>` is contained within, or against the `<ds:Signature>` itself if it is contained within the `<SignatureObject>` element. In these latter two cases, the `<Schema>` element of the input document or `<SignatureObject>` will be used, if present, to identify ID attributes when evaluating the XPointer expression.

If the `<ds:Reference>` uses a full URI (i.e. not a null URI) and the corresponding input document is not present, the server will skip the `<ds:Reference>`, and later return a result code such as `ValidSignature_NotAllDocuments` to indicate this.

2. If the input document is a `<DocumentHash>`, the server checks that the `<ds:Transforms>`, `<ds:DigestMethod>`, and `<ds:DigestValue>` elements match between the `<DocumentHash>` and the `<ds:Reference>`.
3. If the input document is a `<Document>` or an XML element selected by evaluating a `<ds:Reference>`'s XPointer expression, the server applies any transforms specified by the `<ds:Reference>` that have have not already been applied to the input document, and then hashes the resultant data object according to `<ds:DigestMethod>`, and checks that the result matches `<ds:DigestValue>`.
4. The server then validates the signature according to section 3.2.2 in **[XMLSig]**.
5. If the signature validates correctly, the server returns one of the first three `<ResultMinor>` codes listed in section 4.4, depending on the relationship of the signature to the input documents (not including the relationship of the signature to those XML elements that were resolved through XPointer evaluation; the client will have to inspect those relationships manually). If the signature fails to validate correctly, the server returns some other code; either one defined in section 4.4 of this specification, or one defined by some profile of this specification.

4.3.1 Multi-Signature Verification

If a client requests verification of an entire input document, either using a `<SignaturePtr>` without an `<XPath>` or a missing `<SignaturePtr>` (see section 4.3.1, step 1), then the server MUST determine whether the input document contains zero, one, or more than one `<ds:Signature>` elements. If zero, the server should return a `<ResultMajor>` code of `RequesterError`.

If more than one `<ds:Signature>` elements are present, the server MUST either reject the request with a `<ResultMajor>` code of `RequesterError` and a `<ResultMinor>` code of `NotSupported`, or accept the request and try to verify all of the signatures.

If the server accepts the request in the multi-signature case (or if only a single signature is present) and one of the signatures fails to verify, the server should return one of the error codes in section 4.4, reflecting the first error encountered.

If all of the signatures verify correctly, the server should return the `Success` `<ResultMajor>` code and the following `<ResultMinor>` code:

`urn:oasis:names:tc:dss:1.0:resultminor:ValidMultiSignatures`

Upong receiving this result code, the client SHOULD NOT assume any particular relationship between the signature and the input document(s). To check such a relationship, the client would have to verify or inspect the signatures individually.

Only certain optional inputs and outputs are allowed when performing multi-signature verification. See section 4.6 for details.

4.4 Result Codes

Whether the signature succeeds or fails to verify, the server will return the `Success` `<ResultMajor>` code. The `<ResultMinor>` URI MUST be one of the following values, or some other value defined by some profile of this specification. The first three values listed below indicate that the signature or timestamp is valid. Any other value SHALL signal an error of some sort.

`urn:oasis:names:tc:dss:1.0:resultminor:ValidSignature_OnAllDocuments`

The signature or timestamp is valid. Furthermore, the signature or timestamp covers all of the input documents just as they were passed in by the client.

`urn:oasis:names:tc:dss:1.0:resultminor:ValidSignature_OnTransformedDocuments`

The signature or timestamp is valid. Furthermore, the signature or timestamp covers all of the input documents. However, some or all of the input documents have additional transforms applied to them that were not specified by the client.

`urn:oasis:names:tc:dss:1.0:resultminor:ValidSignature_NotAllDocuments`

The signature or timestamp is valid. However, the signature or timestamp does not cover all of the input documents that were passed in by the client.

`urn:oasis:names:tc:dss:1.0:resultminor:IndeterminateKey`

The server could not determine whether the signing key is valid. For example, the server might not have been able to construct a certificate path to the signing key.

`urn:oasis:names:tc:dss:1.0:resultminor:UntrustedKey`

The signature is performed by a key the server considers suspect. For example, the signing key may have been revoked, or it may be a different key from what the server is expecting the signer to use.

`urn:oasis:names:tc:dss:1.0:resultminor:IncorrectSignature`

The signature fails to verify, indicating that the message was modified in transit, or that the signature was performed incorrectly.

`urn:oasis:names:tc:dss:1.0:resultminor:InappropriateSignature`

The signature or its contents are not appropriate in the current context. For example, the signature may be associated with a signature policy and semantics which the DSS server considers unsatisfactory.

4.5 Basic Processing for CMS Signatures

A DSS server that verifies CMS signatures SHOULD perform the following steps, upon receiving a `<VerifyRequest>`. These steps may be changed or overridden by the optional inputs, or by the profile or policy the server is operating under.

1. The server retrieves the CMS signature by decoding the `<Base64Signature>` child of `<SignatureObject>`.
2. The server retrieves the input data. If the CMS signature is detached, there must be a single input document: i.e. a single `<Document>` or `<DocumentHash>` element. Otherwise, if the CMS signature is enveloping, it contains its own input data and there MUST NOT be any input documents present.

3. The CMS signature and input data are verified in the conventional way (see **[RFC 3369]** for details).

4. If the signature validates correctly, the server returns the first `<ResultMinor>` code listed in section 4.4. If the signature fails to validate correctly, the server returns some other code; either one defined in section 4.4 of this specification, or one defined by some profile of this specification.

4.6 Optional Inputs and Outputs

This section defines some optional inputs and outputs that profiles of the DSS verifying protocol might find useful. Section 2.8 defines some common optional inputs that can also be used with the verifying protocol. Profiles of the verifying protocol can define their own optional inputs and outputs, as well. General handling of optional inputs and outputs is discussed in section 2.7.

4.6.1 Optional Input <VerifyManifests>

The presence of this element instructs the server to attempt to validate any input documents it encounters whose `Type` attribute equals `http://www.w3.org/2000/09/xmldsig#Manifest`. Such an input document MUST contain an XML element of type **ds:ManifestType**. On encountering such a document in step 2 of basic processing, the server should repeat step 2 for all the `<ds:Reference>` elements within the manifest. If an error occurs while verifying a `<ds:Reference>` within a manifest, it will be treated no differently from an error that occurs while verifying a `<ds:Reference>` within a `<ds:SignedInfo>`.

This optional input is allowed in multi-signature verification.

```
<xs:element name="VerifyManifests"/>
```

4.6.2 Optional Input <VerificationTime>

This element instructs the server to attempt to determine the signature's validity at the specified time, instead of the current time.

This optional input is allowed in multi-signature verification.

```
<xs:element name="VerificationTime" type="xs:dateTime"/>
```

4.6.3 Optional Input <AdditionalKeyInfo>

This element provides the server with additional data (such as certificates and CRLs) which it can use to validate the signing key.

This optional input is not allowed in multi-signature verification.

```
<xs:element name="AdditionalKeyInfo">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="ds:KeyInfo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

4.6.4 Optional Input <ReturnProcessingDetails> and Output <ProcessingDetails>

The presence of the `<ReturnProcessingDetails>` optional input instructs the server to return a `<ProcessingDetails>` output.

These options are not allowed in multi-signature verification.

```
<xs:element name="ReturnProcessingDetails"/>
```

The `<ProcessingDetails>` optional output elaborates on what signature verification steps succeeded or failed. It may contain the following child elements:

`<ValidDetail>` [Any Number]

A verification detail that was evaluated and found to be valid.

`<IndeterminateDetail>` [Any Number]

A verification detail that could not be evaluated or was evaluated and returned an indeterminate result.

`<InvalidDetail>` [Any Number]

A verification detail that was evaluated and found to be invalid.

```
<xs:element name="ProcessingDetails">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ValidDetail" type="dss:DetailType"
                        minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="IndeterminateDetail"
                        type="dss:DetailType"
                        minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="InvalidDetail" type="xs:dss:DetailType"
                        minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Each detail element is of type **dss:DetailType**. A **dss:DetailType** contains the following child elements and attributes:

`Type` [Required]

A URI which identifies the detail. It may be a value defined by this specification, or a value defined by some other specification. For the values defined by this specification, see below.

Multiple detail elements of the same `Type` may appear in a single `<ProcessingDetails>`. For example, when a signature contains a certificate chain that certifies the signing key, there may be details of the same `Type` present for each certificate in the chain, describing how each certificate was processed.

`<Code>` [Optional]

A URI which more precisely specifies why this detail is valid, invalid, or indeterminate. It must be a value defined by some other specification, since this specification defines no values for this element.

`<Message>` [Optional]

A human-readable message which MAY be logged, used for debugging, etc.

```
<xs:complexType name="DetailType">
    <xs:sequence>
        <xs:element name="Code" type="xs:anyURI" minOccurs="0"/>
        <xs:element name="Message" type="InternationalStringType"
                    minOccurs="0"/>
        <xs:any processContents="lax" minOccurs="0"
                maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Type" type="xs:anyURI" use="required"/>
</xs:element>
```

The values for the `Type` attribute defined by this specification are the following:

`urn:oasis:names:tc:dss:1.0:detail:IssuerTrust`

Whether the issuer of trust information for the signing key (or one of the certifying keys) is considered to be trustworthy.

`urn:oasis:names:tc:dss:1.0:detail:RevocationStatus`

Whether the trust information for the signing key (or one of the certifying keys) is revoked.

`urn:oasis:names:tc:dss:1.0:detail:ValidityInterval`

Whether the trust information for the signing key (or one of the certifying keys) is within its validity interval.

`urn:oasis:names:tc:dss:1.0:detail:Signature`

Whether the document signature (or one of the certifying signatures) verifies correctly.

4.6.5 Optional Input <ReturnSigningTime> and Output <SigningTime>

The presence of the `<ReturnSigningTime>` optional input instructs the server to return a `<SigningTime>` output. This output typically gives the client access to a time value carried within a signature attribute or a signature timestamp, or within a timestamp token if the signature itself is a timestamp (e.g. see section 5.1.1). If no such value is present, and the server has no other way of determining when the signature was performed, the server should omit the `<SigningTime>` output. If there are multiple such values present, behavior is profile-defined.

These options are not allowed in multi-signature verification.

```
<xs:element name="ReturnSigningTime"/>
```

The `<SigningTime>` optional output contains an indication of when the signature was performed, and a boolean attribute that indicates whether this value is attested to by a third-party timestamp authority (if true), or only by the signer (if false).

```
<xs:element name="SigningTime">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:dateTime">
                <xs:attribute name="ThirdPartyTimestamp"
                              type="xs:boolean" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
```

4.6.6 Optional Input <ReturnSignerIdentity> and Output <SignerIdentity>

The presence of the `<ReturnSignerIdentity>` optional input instructs the server to return a `<SignerIdentity>` output.

This optional input and output are not allowed in multi-signature verification.

```
<xs:element name="ReturnSignerIdentity"/>
```

The `<SignerIdentity>` optional output contains an indication of who performed the signature.

```
<xs:element name="SignerIdentity" type="saml:NameIdentifierType"/>
```

4.6.7 Optional Input <ReturnUpdatedSignature> and Output <UpdatedSignature>

The presence of the `<ReturnUpdatedSignature>` optional input instructs the server to return an `<UpdatedSignature>` output, containing a new or updated signature.

The `Type` attribute on `<ReturnUpdatedSignature>`, if present, defines exactly what it means to “update” a signature. For example, the updated signature may be the original signature with some additional unsigned signature properties added to it (such as timestamps, counter-signatures, or additional information for use in verification), or the updated signature could be an entirely new signature calculated on the same input documents as the input signature. Profiles that use this optional input MUST define the allowed values and their semantics, and the default value, for the `Type` attribute (unless only a single type of updated signature is supported, in which case the `Type` attribute can be omitted).

Multiple occurrences of this optional input can be present in a single verify request message. If multiple occurrences are present, each occurrence MUST have a different `Type` attribute. Each occurrence will generate a corresponding optional output. These optional outputs SHALL be distinguishable based on their `Type` attribute, which will match each output with an input.

These options are not allowed in multi-signature verification.

```
<xs:element name="ReturnUpdatedSignature">
    <xs:complexType>
        <xs:attribute name="Type" type="xs:anyURI" use="optional"/>
    </xs:complexType>
</xs:element>
```

The `<UpdatedSignature>` optional output contains the returned signature.

```
<xs:element name="UpdatedSignature">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="dss:SignatureObject">
        <xs:sequence>
        <xs:attribute name="Type" type="xs:anyURI" use="optional"/>
    </xs:complexType>
</xs:element>
```

4.6.8 Optional Input <ReturnTransformedDocument> and Output <TransformedDocument>

The `<ReturnTransformedDocument>` optional input instructs the server to return an input document to which the XML signature transforms specified by a particular `<ds:Reference>` have been applied. The `<ds:Reference>` is indicated by the zero-based `WhichReference` attribute (0 means the first `<ds:Reference>` in the signature, 1 means the second, and so on). Multiple occurrences of this optional input can be present in a single verify request message. Each occurrence will generate a corresponding optional output.

These options are not allowed in multi-signature verification.

```
<xs:element name="ReturnTransformedDocument">
    <xs:complexType>
        <xs:attribute name="WhichReference" type="xs:integer"
                      use="required"/>
    </xs:complexType>
</xs:element>
```

The `<TransformedDocument>` optional output contains a document corresponding to the specified `<ds:Reference>`, after all the transforms in the reference have been applied. In other words, the hash value of the returned document should equal the `<ds:Reference>` element's `<ds:DigestValue>`. To match outputs to inputs, each `<TransformedDocument>` will contain a `WhichReference` attribute which matches the corresponding optional input.

```
<xs:element name="TransformedDocument">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="dss:Document">
        </xs:sequence>
    </xs:complexType>
    <xs:attribute name="WhichReference" type="xs:integer"
                  use="required"/>
</xs:element>
```

5 DSS Core Elements

This section defines two XML elements that may be used in conjunction with the DSS core protocols.

5.1 Element <Timestamp>

This section defines an XML timestamp. A `<Timestamp>` contains some type of timestamp token, such as an RFC 3161 `TimeStampToken` **[RFC 3161]** or a `<ds:Signature>` (aka an "XML timestamp token"). Profiles may introduce additional types of timestamp tokens. XML timestamps can be produced and verified using the timestamping profile of the DSS core protocols **[XML-TSP]**.

An XML timestamp may contain:

`<ds:Signature>` [Optional]

This is an enveloping XML signature, as defined in section 5.1.1.

`<RFC3161TimeStampToken>` [Optional]

This is a base64-encoded `TimeStampToken` as defined in **[RFC3161]**.

```
<xs:element name="Timestamp">
    <xs:complexType>
        <xs:choice>
            <xs:element ref="ds:Signature"/>
            <xs:element name="RFC3161TimeStampToken"
                        type="xs:base64Binary"/>
            <xs:any processContents="lax"/>
        <xs:choice>
    </xs:complexType>
</xs:element>
```

5.1.1 XML Timestamp Token

An XML timestamp token is similar to an RFC 3161 `TimeStampToken`, but is encoded as a `<TstInfo>` element (see section 5.1.2) inside an enveloping `<ds:Signature>`. This allows conventional XML signature implementations to validate the signature, though additional processing is still required to validate the timestamp properties (see section 5.1.3).

The following text describes how the child elements of the `<ds:Signature>` MUST be used:

`<ds:KeyInfo>` [Required]

The `<ds:KeyInfo>` element SHALL identify the issuer of the timestamp and MAY be used to locate, retrieve and validate the timestamp token signature-verification key. The exact details of this element may be specified further in a profile.

`<ds:SignedInfo>/<ds:Reference>` [Required]

There MUST be a single `<ds:Reference>` element whose `URI` attribute references the `<ds:Object>` containing the enveloped `<TstInfo>` element, and whose `Type` attribute is equal to `urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken`. The remaining `<ds:Reference>` element(s) will reference the document or documents that are timestamped.

`<ds:Object>` [Required]

A `<TstInfo>` element SHALL be contained in a `<ds:Object>` element.

5.1.2 Element <TstInfo>

A `<TstInfo>` element is included in an XML timestamp token as a `<ds:Signature>/<ds:Object>` child element. A `<TstInfo>` element has the following children:

`<SerialNumber>` [Required]

This element SHALL contain a serial number produced by the timestamp authority (TSA). It MUST be unique across all the tokens issued by a particular TSA.

`<CreationTime>` [Required]

The time at which the token was issued.

`<Policy>` [Optional]

This element SHALL identify the policy under which the token was issued. The TSA's policy SHOULD identify the fundamental source of its time.

`<ErrorBound>` [Optional]

The TSA's estimate of the maximum error in its local clock.

`<Ordered>` [Default="false"]

This element SHALL indicate whether or not timestamps issued by this TSA, under this policy, are strictly ordered according to the value of the `CreationTime` element value.

TSA [Optional]

The name of the TSA.

```
<xs:element name="TstInfo">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="SerialNumber" type="xs:integer"/>
            <xs:element name="CreationTime" type="xs:dateTime"/>
            <xs:element name="Policy" type="xs:anyURI" minOccurs="0"/>
            <xs:element name="ErrorBound" type="xs:duration"
                        minOccurs="0"/>
            <xs:element name="Ordered" type="xs:boolean"
                        default="false" minOccurs="0"/>
            <xs:element name="TSA" type="saml:NameIdentifierType"
                        minOccurs="0"/>
        <xs:sequence>
    </xs:complexType>
</xs:element>
```

5.1.3 Timestamp verification procedure

If any one of these steps results in failure, then the timestamp token SHOULD be rejected.

1. Locate and verify the signature-verification key corresponding to the `ds:KeyInfo/` element contents.
2. Verify that the signature-verification key is authorized for verifying timestamps.

3. Verify that the signature-verification key conforms with all relevant aspects of the relying-party's policy.
4. Verify that all digest and signature algorithms conform with the relying-party's policy.
5. Verify that the signature-verification key is consistent with the `ds:SignedInfo/SignatureMethod/@Algorithm` element value.
6. Verify that there is a single `ds:SignedInfo/Reference` element whose URI attribute references a `<ds:Object>` containing an enveloped `<TstInfo>` element.
7. Verify that there is a `ds:SignedInfo/Reference` element whose URI attribute correctly identifies the timestamped document.
8. Verify that the `tstInfo/Policy` element value is acceptable.
9. Verify all digests and the signature.

If comparing the `tstInfo/CreationTime` element value to another time value, first verify that they differ by more than the error bound value.

5.2 Element <RequesterIdentity>

This section contains the definition of an XML Requester Identity element. This element can be used as a signature property in an XML signature to identify the client who requested the signature.

This element has the following children:

`Name` [Required]

The name or role of the requester who requested the signature be performed.

`SupportingInfo` [Optional]

Information supporting the name (such as a SAML Assertion **[SAMLCore1.1]**, Liberty Alliance Authentication Context, or X.509 Certificate).

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

```
<xs:element name="RequesterIdentity">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Name" type="saml:NameIdentifierType"/>
            <xs:element name="SupportingInfo" type="dss:AnyType"
                        minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

6 DSS Core Bindings

Mappings from DSS messages into standard communications protocols are called DSS *bindings*. *Transport bindings* specify how DSS messages are encoded and carried over some lower-level transport protocol. *Security bindings* specify how confidentiality, authentication, and integrity can be achieved for DSS messages in the context of some transport binding.

Below we specify an initial set of bindings for DSS. Future bindings may be introduced by the OASIS DSS TC or by other parties.

6.1 HTTP POST Transport Binding

In this binding, the DSS request/response exchange occurs within an HTTP POST exchange **[RFC 2616]**. The following rules apply to the HTTP request:

1. The client may send an HTTP/1.0 or HTTP/1.1 request.
2. The Request URI may be used to indicate a particular service endpoint.
3. The `Content-Type` header MUST be set to “application/xml”.
4. The `Content-Length` header MUST be present and correct.
5. The DSS request message MUST be sent in the body of the HTTP Request.

The following rules apply to the HTTP Response:

3. The `Content-Type` header MUST be set to “text/xml”.
4. The `Content-Length` header MUST be present and correct.
5. The DSS response message MUST be sent in the body of the HTTP Response.
6. The HTTP status code MUST be set to 200 if a DSS response message is returned. Otherwise, the status code can be set to 3*xx* to indicate a redirection, 4*xx* to indicate a low-level client error (such as a malformed request), or 5*xx* to indicate a low-level server error.

6.2 SOAP 1.2 Transport Binding

In this binding, the DSS request/response exchange occurs using the SOAP 1.2 message protocol **[SOAP]**. The following rules apply to the SOAP request:

1. A single DSS `<SignRequest>` or `<VerifyRequest>` element will be transmitted within the body of the SOAP message.
2. The client MUST NOT include any additional XML elements in the SOAP body.
3. The UTF-8 character encoding must be used for the SOAP message.
4. Arbitrary SOAP headers may be present.

The following rules apply to the SOAP response:

1. The server MUST return either a single DSS `<SignResponse>` or `<VerifyResponse>` element within the body of the SOAP message, or a SOAP fault code.
2. The server MUST NOT include any additional XML elements in the SOAP body.
3. If a DSS server cannot parse a DSS request, or there is some error with the SOAP envelope, the server MUST return a SOAP fault code. Otherwise, a DSS result code should be used to signal errors.

4. The UTF-8 character encoding must be used for the SOAP message.
5. Arbitrary SOAP headers may be present.
6. On receiving a DSS response in a SOAP message, the client MUST NOT send a fault code to the DSS server.

6.3 TLS Security Bindings

TLS **[RFC 2246]** is a session-security protocol that can provide confidentiality, authentication, and integrity to the HTTP POST transport binding, the SOAP 1.2 transport binding, or others. TLS supports a variety of authentication methods, so we define several security bindings below. All of these bindings inherit the following rules:

1. TLS 1.0 MUST be supported. SSL 3.0 MAY be supported. Future versions of TLS MAY be supported.
2. RSA ciphersuites MUST be supported. Diffie-Hellman and DSS ciphersuites MAY be supported.
3. TripleDES ciphersuites MUST be supported. AES ciphersuites SHOULD be supported. Other ciphersuites MAY be supported, except for weak ciphersuites intended to meet export restrictions, which SHOULD NOT be supported.

6.3.1 TLS X.509 Server Authentication

The following ciphersuites defined in **[RFC 2246]** and **[RFC 3268]** are supported. The server MUST authenticate itself with an X.509 certificate chain **[RFC 3280]**. The server MUST NOT request client authentication.

MUST:

TLS_RSA_WITH_3DES_EDE_CBC_SHA

SHOULD:

TLS_RSA_WITH_AES_128_CBC_SHA

TLS_RSA_WITH_AES_256_CBC_SHA

6.3.2 TLS X.509 Mutual Authentication

The same ciphersuites mentioned in section 6.2.1 are supported. The server MUST authenticate itself with an X.509 certificate chain, and MUST request client authentication. The client MUST authenticate itself with an X.509 certificate chain.

6.3.3 TLS SRP Authentication

SRP is a way of using a username and password to accomplish mutual authentication. The following ciphersuites defined in **[draft-ietf-tls-srp-08]** are supported.

MUST:

TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA

SHOULD:

TLS_SRP_SHA_WITH_AES_128_CBC_SHA

TLS_SRP_SHA_WITH_AES_256_CBC_SHA

6.3.4 TLS SRP and X.509 Server Authentication

SRP can be combined with X.509 server authentication. The following ciphersuites defined in **[draft-ietf-tls-srp-08]** are supported.

MUST:

TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA

SHOULD:

TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA

TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA

7 DSS-Defined Identifiers

The following sections define various URI-based identifiers. Where possible an existing URN is used to specify a protocol. In the case of IETF protocols the URN of the most current RFC that specifies the protocol is used (see **[RFC 2648]**). URI references created specifically for DSS have the following stem:

urn:oasis:names:tc:dss:1.0:

7.1 Signature Type Identifiers

The following identifiers MAY be used as the content of the `<SignatureType>` optional input (see section 3.5.1).

7.1.1 XML Signature

URI: urn:ietf:rfc:3275

This refers to an XML signature per **[XMLSig]**.

7.1.2 XML TimeStampToken

URI: urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken

This refers to an XML timestamp containing an XML signature, per section 5.1.

7.1.3 RFC 3161 TimeStampToken

URI: urn:ietf:rfc:3161

This refers to an XML timestamp containing an ASN.1 TimeStampToken, per **[RFC 3161]**.

7.1.4 CMS Signature

URI: urn:ietf:rfc:3369

This refers to a CMS signature per **[RFC 3369]**.

7.1.5 PGP Signature

URI: urn:ietf:rfc:2440

This refers to a PGP signature per **[RFC 2440]**.

8 Editorial Issues

1) Another way of handling the options is to have each option placed within an `<Option>` element. This has the advantage that each option could be tagged with a `mustUnderstand` attribute, so the server would know whether it was okay to ignore the option or not. It has the disadvantage of making things a little more verbose.

   Resolution: Leave as is, per 10/20/2003 meeting.

2) It is suggested that the RequestID option be put in the top level of the protocol structure so that it can be used at the basic level of the DSS protocol handler.

   Resolution: This has been done, per 10/20/2003 meeting.

3) The utility of the <DocumentURI> element has been questioned.

   Resolution: Since Rich, John, Trevor, and perhaps Andreas seem in favor of removing this, and only Gregor and Juan Carlos, and perhaps Nick, seem in favor of keeping it, it's been removed.

4) Should every Output only be returned if the client requests it, through an Option?

   Resolution: No – Servers can return outputs on their own initiative, per 11/3/2003 meeting.

5) Should Signature Placement, and elements to envelope, be made Signature Options?

   Resolution: Yes – per 11/3/2003 meeting, but hasn't been done yet.

6) Should <Options> be renamed? To <AdditionalInputs>, <Inputs>, <Parameters>, or something else?

   Resolution: Yes - <OptionalInputs> and <OptionalOutputs>

7) Should we adopt a Timestamp more like Dimitri's <Tst>?

   Resolution: No – instead add a <dss:Timestamp> element, per Nick's suggestion on list

8) The <ProcessingDetails> are a little sketchy, these could be fleshed out.

   Resolution: Done – per draft 10, based on list discussions.

9) A <dss:SignatureObject> can contain a <dss:SignaturePtr>, which uses an XPath expression to point to a signature. This allows a client to send an <InputDocument> to the server with an embedded signature, and just point to the signature, without copying it. Is it acceptable to require all servers to support XPath, for this?

   Resolution: This is not only allowed but required when sending enveloped signatures to the server, so the server knows how to apply the enveloped signature transform. This is disallowed when the server returns signatures to the client, cause the bandwidth savings aren't worth the complexity.

10) **NOTE**: This document may be updated as we work on DSS profiles. In particular, we may add additional Signature Types, Timestamp Types, and Updated Signature Types to section 6. We may also add additional optional inputs and outputs, if commonality is discovered across multiple profiles.

11) Should <ServicePolicy> be made a permanent part of the protocols? (i.e. *not* an optional input?)

    Resolution: Yes, added to the Request in wd-13.

12) Should we use URLs or URNs for our schema namespace URI?

    Resolution: URL (in draft 17)

13) Should we add a WSS Security Binding?

    Resolution: not now

14) Should we add some way for an external policy authority to vouch for some portion of a request?

    Resolution: not in the core

15) Should RequestID be removed?

    Resolution: No.

16) Should input documents have a RefId attribute?

    Resolution: No.

17) Should <SignaturePtr> be optional when there's only 1 input doc, with 1 signature?

    Resolution: Yes.

18) Should the server return the <Profile> it used?

    Resolution: Yes.

9 References

9.1 Normative

[Core-XSD] T. Perrin et al. *DSS Schema*. OASIS, **(MONTH/YEAR TBD)**

[RFC 2119] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels.* IETF RFC 2396, August 1998. http://www.ietf.org/rfc/rfc2396.txt.

[RFC 2246] T Dierks, C. Allen. *The TLS Protocol Version 1.0.* IETF RFC 2246, January 1999. http://www.ietf.org/rfc/rfc2246.txt.

[RFC 2396] T. Berners-Lee et al. *Uniform Resource Identifiers (URI): Generic Syntax.* IETF RFC 2396, August 1998. http://www.ietf.org/rfc/rfc2396.txt.

[RFC 2440] J. Callas, L. Donnerhacke, H. Finney, R. Thayer. *OpenPGP Message Format*. IETF RFC 2440, November 1998. http://www.ietf.org/rfc/rfc2440.txt.

[RFC 2616] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC 2616, June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[RFC 2648] R. Moats. *A URN Namespace for IETF Documents*. IETF RFC 2648, August 1999. http://www.ietf.org/rfc/rfc2648.txt.

[RFC 2822] P. Resnick. *Internet Message Format*. IETF RFC 2822, April 2001. http://www.ietf.org/rfc/rfc2822.txt

[RFC 3161] C. Adams, P. Cain, D. Pinkas, R. Zuccherato. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. IETF RFC 3161, August 2001. http://www.ietf.org/rfc/rfc3161.txt.

[RFC 3268] P. Chown. *AES Ciphersuites for TLS*. IETF RFC 3268, June 2002. http://www.ietf.org/rfc/rfc3268.txt.

[RFC 3280] R. Housley, W. Polk, W. Ford, D. Solo. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. IETF RFC 3280, April 2002. http://www.ietf.org/rfc/rfc3280.txt.

[RFC 3369] R. Housley. *Cryptographic Message Syntax*. IETF RFC 3369, August 2002. http://www.ietf.org/rfc/rfc2459.txt.

[SAMLCore1.1] E. Maler et al. *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V 1.1*. OASIS, November 2002. http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf

[Schema1] H. S. Thompson et al. *XML Schema Part 1: Structures.* W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-1/

[SOAP] M. Gudgin et al. *SOAP Version 1.2 Part 1: Messaging Framework.* W3C Recommendation, June 2003. http://www.w3.org/TR/xmlschema-1/

[XML-C14N] J. Boyer. *Canonical XML Version 1.0*. W3C Recommendation, March 2001. http://www.w3.org/TR/xml-c14n

[XML-ns]	T. Bray, D. Hollander, A. Layman. *Namespaces in XML.* W3C Recommendation, January 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114
[XMLSig]	D. Eastlake et al. *XML-Signature Syntax and Processing.* W3C Recommendation, February 2002. http://www.w3.org/TR/1999/REC-xml-names-19990114
[XML-TSP]	T. Perrin et al. *XML Timestamping Profile of the OASIS Digital Signature Services.* W3C Recommendation, February 2002. OASIS, **(MONTH/YEAR TBD)**

1614 Appendix A. Revision History

Rev	Date	By Whom	What
wd-01	2003-10-03	Trevor Perrin	Initial version
wd-02	2003-10-13	Trevor Perrin	Skeleton of verify as well
wd-03	2003-10-19	Trevor Perrin	Added TimeStampToken, References
wd-04	2003-10-29	Trevor Perrin	Fleshed things out
wd-05	2003-11-9	Trevor Perrin	Added Name, clarified options-handling
wd-06	2003-11-12	Trevor Perrin	Added more options/outputs
wd-07	2003-11-25	Trevor Perrin	URNs, <Timestamp>, other changes.
Wd-08	2003-12-6	Trevor Perrin	Many suggestions from Juan Carlos, Frederick, and Nick incorporated.
Wd-09	2004-1-6	Trevor Perrin	A few minor tweaks to fix a typo, add clarity, and change the order of SignResponse's children
wd-10	2004-1-20	Trevor Perrin	Organized references, updated processing details, touched up a few things.
Wd-11	2004-2-04	Trevor Perrin	Added transport and security bindings, and <Language> optional input
wd-12	2004-2-12	Trevor Perrin	Editorial suggestions from Frederick
wd-13	2004-2-29	Trevor Perrin	Added SOAP Transport binding, and made 'Profile' attribute part of the Request messages, instead of an option.
Wd-14	2004-3-07	Trevor Perrin	Fixes from Krishna
wd-15	2004-3-08	Trevor Perrin	Property URI -> QNames, added some Editorial issues
wd-16	2004-3-21	Trevor Perrin	Replaced dss:NameType with saml:NameIdentifierType, per Nick's suggestion.
Wd-17	2004-4-02	Trevor Perrin	Schema URN -> URL, TryAgainLater
wd-18	2004-4-04	Trevor Perrin	Fixes from Karel Wouters
wd-19	2004-4-15	Trevor Perrin	ResultMajor URIs, AdditionalProfile
wd-20	2004-4-19	Trevor Perrin	Updated <Timestamp>, few tweaks

Rev	Date	By Whom	What
wd-21	2004-5-11	Trevor Perrin	CMS, special handling of enveloping/enveloped DSIG, multi-signature DSIG verification.
Wd-23	2004-6-08	Trevor Perrin	Added DTD example, added returned Profile attribute on SignResponse and VerifyResponse.
Wd-24	2004-6-20	Trevor Perrin	Removed xmlns:xml from schema.
Wd-25	2004-6-22	Trevor Perrin	Fixed a typo.
Wd-26	2004-6-28	Trevor Perrin	Mentioned as committee draft
wd-27	200410-04	Trevor Perrin	Gregor Karlinger's feedback
wd-28	200410-18	Trevor Perrin	Added a little text to clarify manifests and <ReturnSigningTime>
wd-29	200411-01	Trevor Perrin	Added a little text to clarify <ReturnUpdatedSignature>, and added <SupportingInfo> to <ClaimedIdentity>

Appendix B. Notices