



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 Kasselmann, 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

39	1	Introduction	4
40	1.1	Notation	4
41	1.2	Schema Organization and Namespaces	4
42	1.3	DSS Overview (Non-normative).....	5
43	2	Common Protocol Structures.....	6
44	2.1	Type AnyType	6
45	2.2	Type InternationalStringType	6
46	2.3	Type saml:NameIdentifierType	6
47	2.4	Element <InputDocuments>.....	6
48	2.4.1	Type DocumentBaseType	7
49	2.4.2	Element <Document>	8
50	2.4.3	Element <DocumentHash>.....	8
51	2.5	Element <SignatureObject>	9
52	2.6	Element <Result>.....	10
53	2.7	Elements <OptionalInputs> and <OptionalOutputs>	11
54	2.8	Common Optional Inputs	12
55	2.8.1	Optional Input <ServicePolicy>.....	12
56	2.8.2	Optional Input <ClaimedIdentity>	12
57	2.8.3	Optional Input <Language>	13
58	2.8.4	Optional Input <AdditionalProfile>	13
59	3	The DSS Signing Protocol	14
60	3.1	Element <SignRequest>	14
61	3.2	Element <SignResponse>	14
62	3.3	Basic Processing for XML Signatures.....	15
63	3.3.1	Enveloping Signatures.....	16
64	3.3.2	Enveloped Signatures.....	16
65	3.4	Basic Processing for CMS Signatures	16
66	3.5	Optional Inputs and Outputs	17
67	3.5.1	Optional Input <SignatureType>	17
68	3.5.2	Optional Input <AddTimestamp>	17
69	3.5.3	Optional Input <IntendedAudience>	17
70	3.5.4	Optional Input <KeySelector>.....	18
71	3.5.5	Optional Input <SignedReferences>.....	18
72	3.5.6	Optional Input <Properties>.....	19
73	3.5.7	Optional Input <SignaturePlacement> and Output <DocumentWithSignature>	20
74	3.5.8	Optional Input <EnvelopingSignature>.....	21
75	4	The DSS Verifying Protocol	23
76	4.1	Element <VerifyRequest>	23
77	4.2	Element <VerifyResponse>	24

78	4.3 Basic Processing for XML Signatures.....	24
79	4.3.1 Multi-Signature Verification.....	25
80	4.4 Result Codes.....	26
81	4.5 Basic Processing for CMS Signatures.....	26
82	4.6 Optional Inputs and Outputs.....	27
83	4.6.1 Optional Input <VerifyManifests>.....	27
84	4.6.2 Optional Input <VerificationTime>.....	27
85	4.6.3 Optional Input <AdditionalKeyInfo>.....	27
86	4.6.4 Optional Input <ReturnProcessingDetails> and Output <ProcessingDetails>.....	28
87	4.6.5 Optional Input <ReturnSigningTime> and Output <SigningTime>.....	29
88	4.6.6 Optional Input <ReturnSignerIdentity> and Output <SignerIdentity>.....	30
89	4.6.7 Optional Input <ReturnUpdatedSignature> and Output <UpdatedSignature>.....	30
90	4.6.8 Optional Input <ReturnTransformedDocument> and Output <TransformedDocument>	
91	31
92	5 DSS Core Elements.....	32
93	5.1 Element <Timestamp>.....	32
94	5.1.1 XML Timestamp Token.....	32
95	5.1.2 Element <TstInfo>.....	33
96	5.1.3 Timestamp verification procedure.....	33
97	5.2 Element <RequesterIdentity>.....	34
98	6 DSS Core Bindings.....	35
99	6.1 HTTP POST Transport Binding.....	35
100	6.2 SOAP 1.2 Transport Binding.....	35
101	6.3 TLS Security Bindings.....	36
102	6.3.1 TLS X.509 Server Authentication.....	36
103	6.3.2 TLS X.509 Mutual Authentication.....	36
104	6.3.3 TLS SRP Authentication.....	36
105	6.3.4 TLS SRP and X.509 Server Authentication.....	37
106	7 DSS-Defined Identifiers.....	38
107	7.1 Signature Type Identifiers.....	38
108	7.1.1 XML Signature.....	38
109	7.1.2 XML TimeStampToken.....	38
110	7.1.3 RFC 3161 TimeStampToken.....	38
111	7.1.4 CMS Signature.....	38
112	7.1.5 PGP Signature.....	38
113	8 Editorial Issues.....	39
114	9 References.....	41
115	9.1 Normative.....	41
116	Appendix A. Revision History.....	43
117	Appendix B. Notices.....	45
118		

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:

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

166 1.3 DSS Overview (Non-normative)

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

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

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

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

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

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

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

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

206 2 Common Protocol Structures

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

208 2.1 Type AnyType

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

```
210 <xs:complexType name="AnyType" >
211   <xs:sequence>
212     <xs:any processContents="lax" maxOccurs="unbounded" />
213   </xs:sequence>
214 </xs:complexType>
```

215 2.2 Type InternationalStringType

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

```
218 <xs:complexType name="InternationalStringType" >
219   <xs:simpleContent>
220     <xs:extension base="xs:string" >
221       <xs:attribute ref="xml:lang" use="required" >
222     </xs:extension base="xs:string" >
223   </xs:simpleContent>
224 </xs:complexType>
```

225 2.3 Type `saml:NameIdentifierType`

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

229 `NameQualifier` [Optional]

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

232 `Format` [Optional]

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

236 2.4 Element `<InputDocuments>`

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

243

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

245 <Document> [Any Number]

246 An XML document or some other data.

247 <DocumentHash> [Any Number]

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

```
249 <xs:element name="InputDocuments">
250   <xs:complexType>
251     <xs:sequence>
252       <xs:choice minOccurs="1" maxOccurs="unbounded">
253         <xs:element ref="dss:Document"/>
254         <xs:element ref="dss:DocumentHash"/>
255         <xs:any processContents="lax"/>
256       </xs:choice>
257     </xs:sequence>
258   </xs:complexType>
259 </xs:element>
```

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

264 2.4.1 Type DocumentBaseType

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

267 ID [Optional]

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

271 RefURI [Optional]

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

275 RefType [Optional]

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

278 <ds:Transforms> [Optional]

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

284 <Schema> [Optional]

285 This may be used when the document contains XML signatures. It transfers an XML Schema
286 **[Schema1]** which gives the ID attributes of elements within the input document, which may be
287 necessary if the included signatures' <ds:Reference> elements use XPointer expressions.

288 See section 4.3, step 2 for details.

```

289 <xs:complexType name="DocumentBaseType" abstract="true">
290   <xs:sequence>
291     <xs:element ref="ds:Transforms" minOccurs="0"/>
292     <xs:element name="Schema" type="xs:base64Binary"
293       minOccurs="0"/>
294   </xs:sequence>
295   <xs:attribute name="ID" type="xs:ID" use="optional"/>
296   <xs:attribute name="RefURI" type="xs:ID" use="optional"/>
297   <xs:attribute name="RefType" type="xs:ID" use="optional"/>
298 </xs:complexType>

```

299 2.4.2 Element <Document>

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

302 <XMLData> [Optional]

303 This contains arbitrary XML content.

304 <Base64Data> [Optional]

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

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

```

311 <xs:element name="Document" >
312   <xs:complexType>
313     <xs:complexContent>
314       <xs:extension base="dss:DocumentBaseType">
315         <xs:choice>
316           <xs:element ref="dss:XMLData" />
317           <xs:element ref="dss:Base64Data" />
318         </xs:choice>
319       </xs:extension>
320     </xs:complexContent>
321   </xs:complexType>
322 </xs:element>
323
324 <xs:element name="XMLData" type="dss:AnyType" />
325
326 <xs:element name="Base64Data">
327   <xs:complexType>
328     <xs:simpleContent>
329       <xs:extension base="xs:base64Binary">
330         <xs:attribute name="MimeType" type="xs:string"
331           use="optional">
332       </xs:extension>
333     </xs:simpleContent>
334   </xs:complexType>
335 </xs:element>

```

336 2.4.3 Element <DocumentHash>

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

339 <ds:DigestMethod> [Required]
340 This identifies the digest algorithm used to hash the document. This specifies the value for a
341 <ds:Reference> element's <ds:DigestMethod> child element when referring to this input
342 document.

343 <ds:DigestValue> [Required]
344 This gives the document's hash value. This specifies the value for a <ds:Reference>
345 element's <ds:DigestValue> child element when referring to this input document.

```
346 <xs:element name="DocumentHash">  
347   <xs:complexType>  
348     <xs:complexContent>  
349       <xs:extension base="dss:DocumentBaseType">  
350         <xs:sequence>  
351           <xs:element ref="ds:DigestMethod"/>  
352           <xs:element ref="ds:DigestValue"/>  
353         </xs:sequence>  
354       </xs:extension>  
355     </xs:complexContent>  
356   </xs:complexType>  
357 </xs:element>
```

358 2.5 Element <SignatureObject>

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

362 <ds:Signature> [Optional]

363 An XML signature [**XMLSig**].

364 <Timestamp> [Optional]

365 An XML timestamp (see section 5.1).

366 <Base64Signature> [Optional]

367 A base64 encoding of some non-XML signature, such as a PGP [**RFC 2440**] or CMS [**RFC**
368 **3369**] signature. The type of signature is specified by its `Type` attribute (see section 7.1).

369 <SignaturePtr> [Optional]

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

371 Schema [Optional]

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

377 A <SignaturePtr> contains the following attributes:

378 whichDocument [Required]

379 This identifies the input document being pointed at.

380 XPath [Optional]

381 This identifies the element being pointed at. The XPath expression is evaluated from the
382 context node identified by the whichDocument attribute.

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

```
385 <xs:element name="SignatureObject">
386   <xs:complexType>
387     <xs:sequence>
388       <xs:choice>
389         <xs:element ref="ds:Signature"/>
390         <xs:element ref="dss:Timestamp"/>
391         <xs:element ref="dss:Base64Signature"/>
392         <xs:element ref="dss:SignaturePtr"/>
393         <xs:any processContents="lax"/>
394       </xs:choice>
395       <xs:element name="Schema" type="xs:base64Binary"
396         minOccurs="0"/>
397     </xs:sequence>
398   </xs:complexType>
399 </xs:element>
400
401 <xs:element name="Base64Signature">
402   <xs:complexType>
403     <xs:simpleContent>
404       <xs:extension base="xs:base64Binary">
405         <xs:attribute name="Type" type="xs:anyURI"/>
406       </xs:extension>
407     </xs:simpleContent>
408   </xs:complexType>
409 </xs:element>
410
411 <xs:element name="SignaturePtr">
412   <xs:complexType>
413     <xs:attribute name="WhichDocument" type="xs:IDREF"/>
414     <xs:attribute name="XPath" type="xs:string" use="optional"/>
415   </xs:complexType>
416 </xs:element>
```

417 2.6 Element <Result>

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

420 <ResultMajor> [Required]

421 The most significant component of the result code.

422 <ResultMinor> [Optional]

423 The least significant component of the result code.

424 <ResultMessage> [Optional]

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

426

427

428

429

430

431

```

432 <xs:element name="Result">
433   <xs:complexType>
434     <xs:sequence>
435       <xs:element name="ResultMajor" type="xs:anyURI"/>
436       <xs:element name="ResultMinor" type="xs:anyURI"
437         minOccurs="0"/>
438       <xs:element name="ResultMessage"
439         type="InternationalStringType" minOccurs="0"/>
440     </xs:sequence>
441   </xs:complexType>
442 </xs:element>

```

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

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

447 The protocol executed successfully.

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

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

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

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

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

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

455 The client is not authorized to perform the request.

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

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

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

460 2.7 Elements <OptionalInputs> and <OptionalOutputs>

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

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

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

475 The <OptionalInputs> and <OptionalOutputs> elements contain unordered inputs and
 476 outputs. Applications MUST be able to handle optional inputs or outputs appearing in any order

477 within these elements. Normally, there will only be at most one occurrence of any particular
478 optional input or output within a protocol message. Where multiple occurrences of an optional
479 input or optional output are allowed, it will be explicitly specified (see section 4.6.8 for an
480 example).

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

```
483 <xs:element name="OptionalInputs" type="dss:AnyType"/>  
484 <xs:element name="OptionalOutputs" type="dss:AnyType"/>  
485
```

486 2.8 Common Optional Inputs

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

488 2.8.1 Optional Input <ServicePolicy>

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

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

495 2.8.2 Optional Input <ClaimedIdentity>

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

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

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

```
507 <xs:element name="ClaimedIdentity">  
508 <xs:complexType>  
509 <xs:sequence>  
510 <xs:element name="Name" type="saml:NameIdentifierType"/>  
511 <xs:element name="SupportingInfo" type="dss:AnyType"  
512 minOccurs="0"/>  
513 </xs:sequence>  
514 </xs:complexType>  
515 </xs:element>
```

516 **2.8.3 Optional Input <Language>**

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

520

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

521 **2.8.4 Optional Input <AdditionalProfile>**

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

526

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

527

528

529 3 The DSS Signing Protocol

530 3.1 Element <SignRequest>

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

533 RequestID [Optional]

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

536 Profile [Optional]

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

540 <OptionalInputs> [Optional]

541 Any additional inputs to the request.

542 <InputDocuments> [Required]

543 The input documents which the signature will be calculated over.

```
544 <xs:element name="SignRequest">  
545   <xs:complexType>  
546     <xs:sequence>  
547       <xs:element ref="dss:OptionalInputs" minOccurs="0"/>  
548       <xs:element ref="dss:InputDocuments"/>  
549     </xs:sequence>  
550     <xs:attribute name="RequestID" type="xs:string"  
551                 use="optional"/>  
552     <xs:attribute name="Profile" type="xs:anyURI" use="optional"/>  
553   </xs:complexType>  
554 </xs:element>
```

555 3.2 Element <SignResponse>

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

557 RequestID [Optional]

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

560 Profile [Optional]

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

563 <Result> [Required]

564 A code representing the status of the request.

565 <OptionalOutputs> [Optional]

566 Any additional outputs returned by the server.

567 <SignatureObject> [Optional]

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

```
570 <xs:element name="SignResponse">  
571   <xs:complexType>  
572     <xs:sequence>  
573       <xs:element ref="dss:Result" />  
574       <xs:element ref="dss:OptionalOutputs" minOccurs="0" />  
575       <xs:element ref="dss:SignatureObject" minOccurs="0" />  
576     </xs:sequence>  
577     <xs:attribute name="RequestID" type="xs:string"  
578                 use="optional" />  
579     <xs:attribute name="Profile" type="xs:anyURI" use="required" />  
580   </xs:complexType>  
581 </xs:element>
```

582 3.3 Basic Processing for XML Signatures

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

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

- 612 e. The `<ds:Transforms>` element is set to the input document's `<ds:Transforms>`
613 element. If any additional transforms were applied by the server in step 1.c., these
614 are appended as well.
- 615 3. The server creates an XML signature using the `<ds:Reference>` elements created in Step
616 2, according to the processing rules in **[XMLSig]**.

617 3.3.1 Enveloping Signatures

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

621 3.3.2 Enveloped Signatures

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

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

629 3.4 Basic Processing for CMS Signatures

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

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

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

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

656 3.5 Optional Inputs and Outputs

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

661 3.5.1 Optional Input <SignatureType>

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

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

666 3.5.2 Optional Input <AddTimestamp>

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

```
672 <xs:element name="AddTimestamp">  
673   <xs:complexType>  
674     <xs:attribute name="Type" type="xs:anyURI" use="optional" />  
675   </xs:complexType>  
676 </xs:element>
```

677 3.5.3 Optional Input <IntendedAudience>

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

```
681 <xs:element name="IntendedAudience">  
682   <xs:complexType>  
683     <xs:sequence>  
684       <xs:element name="Recipient" type="saml:NameIdentifierType"  
685         maxOccurs="unbounded" />  
686     </xs:sequence>  
687   </xs:complexType>  
688 </xs:element>
```

689

690

691 3.5.4 Optional Input <KeySelector>

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

```
693 <xs:element name="KeySelector">
694   <xs:complexType>
695     <xs:choice>
696       <xs:element ref="ds:KeyInfo" />
697       <xs:any processContents="lax" />
698     </xs:choice>
699   </xs:complexType>
700 </xs:element>
```

701 3.5.5 Optional Input <SignedReferences>

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

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

714 Each <SignedReference> element contains:

715 WhichDocument [Required]

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

717 RefId [Optional]

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

719 <ds:Transforms> [Optional]

720 Requests the server to perform additional transforms on this reference.

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

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

- 734 f. If the end result of these transforms is an XML node set, the server must convert
 735 the node set back into an octet stream using Canonical XML [XML-C14N].
- 736 g. The server hashes the resultant octet stream.
- 737 2. The server forms a <ds:Reference> for each <SignedReference>. The elements
 738 and attributes of the <ds:Reference> are set as follows:
- 739 a. The URI attribute is set to the referenced input document's RefURI attribute. If
 740 the input document has no RefURI attribute, the <ds:Reference> element's
 741 URI attribute is omitted. A signature MUST NOT be created if more than one
 742 RefURI is omitted in the set of input documents, or if more than one
 743 <SignedReference> refers to an input document that has no RefURI.
- 744 b. The Type attribute is set to the referenced input document's RefType attribute.
 745 If the input document has no RefType attribute, the <ds:Reference>
 746 element's Type attribute is omitted.
- 747 c. The Id attribute is set to the <SignedReference> element's RefId attribute.
 748 If the <SignedReference> has no RefId attribute, the <ds:Reference>
 749 element's Id attribute is omitted.
- 750 d. The <ds:DigestMethod> element is set to the hash method that was used in
 751 step 1.g (for a <Document>), or to the referenced input document's
 752 <ds:DigestMethod> (for a <DocumentHash>).
- 753 e. The <ds:DigestValue> element is set to the hash value that was calculated in
 754 step 1.g (for a <Document>), or to the referenced input document's
 755 <ds:DigestValue> (for a <DocumentHash>).
- 756 f. The <ds:Transforms> element is set to the referenced input document's
 757 <ds:Transforms> element with the <SignedReference> element's
 758 transforms appended. If any additional transforms were applied by the server in
 759 step 1.e., these are appended as well.

```

760 <xs:element name="SignedReferences">
761   <xs:complexType>
762     <xs:sequence>
763       <xs:element ref="dss:SignedReference"
764         maxOccurs="unbounded" />
765     </xs:sequence>
766   </xs:complexType>
767 </xs:element>
768
769 <xs:element name="SignedReference">
770   <xs:complexType>
771     <xs:sequence>
772       <xs:element ref="ds:Transforms" minOccurs="0" />
773     </xs:sequence>
774     <xs:attribute name="WhichDocument" type="xs:IDREF"
775       use="required" />
776     <xs:attribute name="RefId" type="xs:string" use="optional" />
777   </xs:complexType>
778 </xs:element>
  
```

779 3.5.6 Optional Input <Properties>

780 The <Properties> element is used to request that the server add certain signed or unsigned
 781 properties (aka "signature attributes") into the signature. The client can send the server a

782 particular value to use for each property, or leave the value up to the server to determine. The
783 server can add additional properties, even if these aren't requested by the client.

784 The <Properties> element contains:

785 <SignedProperties> [Optional]

786 These properties will be covered by the signature.

787 <UnsignedProperties> [Optional]

788 These properties will not be covered by the signature.

789 Each <Property> element contains:

790 <Identifier> [Required]

791 A URI reference identifying the property.

792 <Value> [Optional]

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

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

```
796 <xs:element name="Properties">  
797   <xs:complexType>  
798     <xs:sequence>  
799       <xs:element name="SignedProperties"  
800         type="dss:PropertiesType" minOccurs="0"/>  
801       <xs:element name="UnsignedProperties"  
802         type="dss:PropertiesType" minOccurs="0"/>  
803     </xs:sequence>  
804   </xs:complexType>  
805 </xs:element>  
806  
807 <xs:complexType name="PropertiesType">  
808   <xs:sequence>  
809     <xs:element ref="dss:Property" maxOccurs="unbounded"/>  
810   </xs:sequence>  
811 </xs:complexType>  
812  
813 <xs:element name="Property">  
814   <xs:complexType>  
815     <xs:sequence>  
816       <xs:element name="Identifier" type="xs:anyURI"/>  
817       <xs:element name="Value" type="dss:AnyType"  
818         minOccurs="0"/>  
819     </xs:sequence>  
820   </xs:complexType>  
821 </xs:element>
```

822 3.5.7 Optional Input <SignaturePlacement> and Output 823 <DocumentWithSignature>

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

829 signature in the input document. The <SignaturePlacement> element contains the following
830 attributes and elements:

831 WhichDocument [Required]

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

834 <XpathAfter> [Optional]

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

838 <XpathFirstChildOf> [Optional]

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

```
843 <xs:element name="SignaturePlacement">  
844   <xs:complexType>  
845     <xs:choice>  
846       <xs:element name="XpathAfter" type="xs:string"/>  
847       <xs:element name="XpathFirstChildOf" type="xs:string"/>  
848     </xs:choice>  
849     <xs:attribute name="WhichDocument" type="xs:IDREF"/>  
850   </xs:complexType>  
851 </xs:element>
```

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

854 <Document> [Optional]

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

```
856 <xs:element name="DocumentWithSignature">  
857   <xs:complexType>  
858     <xs:sequence>  
859       <xs:element ref="Document"/>  
860     </xs:sequence>  
861   </xs:complexType>  
862 </xs:element>
```

863 3.5.8 Optional Input <EnvelopingSignature>

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

868 WhichDocument [Required]

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

871 ObjId [Optional]

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

873

```
874 <xs:element name="EnvelopingSignature">
875   <xs:complexType>
876     <xs:attribute name="WhichDocument" type="xs:IDREF"/>
877     <xs:attribute name="ObjId" type="xs:string" use="optional"/>
878   </xs:complexType>
879 </xs:element>
```

880

4 The DSS Verifying Protocol

881

4.1 Element <VerifyRequest>

882

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:

883

884

RequestID [Optional]

885

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

886

887

Profile [Optional]

888

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.

889

890

891

<OptionalInputs> [Optional]

892

Any additional inputs to the request.

893

<SignatureObject> [Optional]

894

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.

895

896

897

898

899

900

901

<InputDocuments> [Optional]

902

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).

903

904

905

```
<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>
```

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920 4.2 Element <VerifyResponse>

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

922 RequestID [Optional]

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

925 Profile [Optional]

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

928 <Result> [Required]

929 A code representing the status of the corresponding request.

930 <OptionalOutputs> [Optional]

931 Any additional outputs returned by the server.

```
932 <xs:element name="VerifyResponse">  
933   <xs:complexType>  
934     <xs:sequence>  
935       <xs:element ref="dss:Result"/>  
936       <xs:element ref="dss:OptionalOutputs" minOccurs="0"/>  
937     </xs:sequence>  
938     <xs:attribute name="RequestID" type="xs:string"  
939       use="optional"/>  
940     <xs:attribute name="Profile" type="xs:anyURI" use="required"/>  
941   </xs:complexType>  
942 </xs:element>
```

943 4.3 Basic Processing for XML Signatures

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

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

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

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

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

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

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

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

991 4.3.1 Multi-Signature Verification

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

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

1000 If the server accepts the request in the multi-signature case (or if only a single signature is
 1001 present) and one of the signatures fails to verify, the server should return one of the error codes
 1002 in section 4.4, reflecting the first error encountered.

1003 If all of the signatures verify correctly, the server should return the Success <ResultMajor>
 1004 code and the following <ResultMinor> code:

1005 urn:oasis:names:tc:dss:1.0:resultminor:ValidMultiSignatures

1006 Upon receiving this result code, the client SHOULD NOT assume any particular relationship
 1007 between the signature and the input document(s). To check such a relationship, the client would
 1008 have to verify or inspect the signatures individually.

1009 Only certain optional inputs and outputs are allowed when performing multi-signature verification.
1010 See section 4.6 for details.

1011 **4.4 Result Codes**

1012 Whether the signature succeeds or fails to verify, the server will return the `Success`
1013 `<ResultMajor>` code. The `<ResultMinor>` URI MUST be one of the following values, or
1014 some other value defined by some profile of this specification. The first three values listed below
1015 indicate that the signature or timestamp is valid. Any other value SHALL signal an error of some
1016 sort.

1017 `urn:oasis:names:tc:dss:1.0:resultminor:ValidSignature_OnAllDocuments`

1018 The signature or timestamp is valid. Furthermore, the signature or timestamp covers all of the
1019 input documents just as they were passed in by the client.

1020 `urn:oasis:names:tc:dss:1.0:resultminor:ValidSignature_OnTransformedDocu`
1021 `ments`

1022 The signature or timestamp is valid. Furthermore, the signature or timestamp covers all of the
1023 input documents. However, some or all of the input documents have additional transforms
1024 applied to them that were not specified by the client.

1025 `urn:oasis:names:tc:dss:1.0:resultminor:ValidSignature_NotAllDocuments`

1026 The signature or timestamp is valid. However, the signature or timestamp does not cover all of
1027 the input documents that were passed in by the client.

1028 `urn:oasis:names:tc:dss:1.0:resultminor:IndeterminateKey`

1029 The server could not determine whether the signing key is valid. For example, the server
1030 might not have been able to construct a certificate path to the signing key.

1031 `urn:oasis:names:tc:dss:1.0:resultminor:UntrustedKey`

1032 The signature is performed by a key the server considers suspect. For example, the signing
1033 key may have been revoked, or it may be a different key from what the server is expecting the
1034 signer to use.

1035 `urn:oasis:names:tc:dss:1.0:resultminor:IncorrectSignature`

1036 The signature fails to verify, indicating that the message was modified in transit, or that the
1037 signature was performed incorrectly.

1038 `urn:oasis:names:tc:dss:1.0:resultminor:InappropriateSignature`

1039 The signature or its contents are not appropriate in the current context. For example, the
1040 signature may be associated with a signature policy and semantics which the DSS server
1041 considers unsatisfactory.

1042 **4.5 Basic Processing for CMS Signatures**

1043 A DSS server that verifies CMS signatures SHOULD perform the following steps, upon receiving
1044 a `<VerifyRequest>`. These steps may be changed or overridden by the optional inputs, or by
1045 the profile or policy the server is operating under.

- 1046 1. The server retrieves the CMS signature by decoding the `<Base64Signature>` child of
1047 `<SignatureObject>`.
- 1048 2. The server retrieves the input data. If the CMS signature is detached, there must be a single
1049 input document: i.e. a single `<Document>` or `<DocumentHash>` element. Otherwise, if the
1050 CMS signature is enveloping, it contains its own input data and there MUST NOT be any
1051 input documents present.

- 1052 3. The CMS signature and input data are verified in the conventional way (see [RFC 3369] for
1053 details).
- 1054 4. If the signature validates correctly, the server returns the first <ResultMinor> code listed in
1055 section 4.4. If the signature fails to validate correctly, the server returns some other code;
1056 either one defined in section 4.4 of this specification, or one defined by some profile of this
1057 specification.

1058 4.6 Optional Inputs and Outputs

1059 This section defines some optional inputs and outputs that profiles of the DSS verifying protocol
1060 might find useful. Section 2.8 defines some common optional inputs that can also be used with
1061 the verifying protocol. Profiles of the verifying protocol can define their own optional inputs and
1062 outputs, as well. General handling of optional inputs and outputs is discussed in section 2.7.

1063 4.6.1 Optional Input <VerifyManifests>

1064 The presence of this element instructs the server to attempt to validate any input documents it
1065 encounters whose Type attribute equals
1066 <http://www.w3.org/2000/09/xmldsig#Manifest>. Such an input document MUST
1067 contain an XML element of type **ds:ManifestType**. On encountering such a document in step 2
1068 of basic processing, the server should repeat step 2 for all the <ds:Reference> elements within
1069 the manifest. If an error occurs while verifying a <ds:Reference> within a manifest, it will be
1070 treated no differently from an error that occurs while verifying a <ds:Reference> within a
1071 <ds:SignedInfo>.

1072 This optional input is allowed in multi-signature verification.

```
1073 <xs:element name="VerifyManifests"/>
```

1074 4.6.2 Optional Input <VerificationTime>

1075 This element instructs the server to attempt to determine the signature's validity at the specified
1076 time, instead of the current time.

1077 This optional input is allowed in multi-signature verification.

```
1078 <xs:element name="VerificationTime" type="xs:dateTime"/>
```

1079 4.6.3 Optional Input <AdditionalKeyInfo>

1080 This element provides the server with additional data (such as certificates and CRLs) which it can
1081 use to validate the signing key.

1082 This optional input is not allowed in multi-signature verification.

```
1083 <xs:element name="AdditionalKeyInfo">  
1084   <xs:complexType>  
1085     <xs:sequence>  
1086       <xs:element ref="ds:KeyInfo"/>  
1087     </xs:sequence>  
1088   </xs:complexType>  
1089 </xs:element>
```

1090 4.6.4 Optional Input <ReturnProcessingDetails> and Output 1091 <ProcessingDetails>

1092 The presence of the <ReturnProcessingDetails> optional input instructs the server to return
1093 a <ProcessingDetails> output.

1094 These options are not allowed in multi-signature verification.

```
1095 <xs:element name="ReturnProcessingDetails"/>
```

1096 The <ProcessingDetails> optional output elaborates on what signature verification steps
1097 succeeded or failed. It may contain the following child elements:

1098 <ValidDetail> [Any Number]

1099 A verification detail that was evaluated and found to be valid.

1100 <IndeterminateDetail> [Any Number]

1101 A verification detail that could not be evaluated or was evaluated and returned an
1102 indeterminate result.

1103 <InvalidDetail> [Any Number]

1104 A verification detail that was evaluated and found to be invalid.

```
1105 <xs:element name="ProcessingDetails">  
1106   <xs:complexType>  
1107     <xs:sequence>  
1108       <xs:element name="ValidDetail" type="dss:DetailType"  
1109         minOccurs="0" maxOccurs="unbounded"/>  
1110       <xs:element name="IndeterminateDetail"  
1111         type="dss:DetailType"  
1112         minOccurs="0" maxOccurs="unbounded"/>  
1113       <xs:element name="InvalidDetail" type="xs:dss:DetailType"  
1114         minOccurs="0" maxOccurs="unbounded"/>  
1115     </xs:sequence>  
1116   </xs:complexType>  
1117 </xs:element>
```

1118 Each detail element is of type **dss:DetailType**. A **dss:DetailType** contains the following child
1119 elements and attributes:

1120 **Type** [Required]

1121 A URI which identifies the detail. It may be a value defined by this specification, or a
1122 value defined by some other specification. For the values defined by this specification,
1123 see below.

1124 Multiple detail elements of the same **Type** may appear in a single
1125 <ProcessingDetails>. For example, when a signature contains a certificate chain
1126 that certifies the signing key, there may be details of the same **Type** present for each
1127 certificate in the chain, describing how each certificate was processed.

1128 <Code> [Optional]

1129 A URI which more precisely specifies why this detail is valid, invalid, or indeterminate.
1130 It must be a value defined by some other specification, since this specification defines
1131 no values for this element.

1132 <Message> [Optional]

1133 A human-readable message which MAY be logged, used for debugging, etc.

1134

```

1135 <xs:complexType name="DetailType">
1136   <xs:sequence>
1137     <xs:element name="Code" type="xs:anyURI" minOccurs="0" />
1138     <xs:element name="Message" type="InternationalStringType"
1139       minOccurs="0" />
1140     <xs:any processContents="lax" minOccurs="0"
1141       maxOccurs="unbounded" />
1142   </xs:sequence>
1143   <xs:attribute name="Type" type="xs:anyURI" use="required" />
1144 </xs:complexType>

```

1145 The values for the Type attribute defined by this specification are the following:

1146 urn:oasis:names:tc:dss:1.0:detail:IssuerTrust

1147 Whether the issuer of trust information for the signing key (or one of the certifying keys) is
 1148 considered to be trustworthy.

1149 urn:oasis:names:tc:dss:1.0:detail:RevocationStatus

1150 Whether the trust information for the signing key (or one of the certifying keys) is revoked.

1151 urn:oasis:names:tc:dss:1.0:detail:ValidityInterval

1152 Whether the trust information for the signing key (or one of the certifying keys) is within its
 1153 validity interval.

1154 urn:oasis:names:tc:dss:1.0:detail:Signature

1155 Whether the document signature (or one of the certifying signatures) verifies correctly.

1156 **4.6.5 Optional Input <ReturnSigningTime> and Output <SigningTime>**

1157 The presence of the <ReturnSigningTime> optional input instructs the server to return a
 1158 <SigningTime> output. This output typically gives the client access to a time value carried
 1159 within a signature attribute or a signature timestamp, or within a timestamp token if the signature
 1160 itself is a timestamp (e.g. see section 5.1.1). If no such value is present, and the server has no
 1161 other way of determining when the signature was performed, the server should omit the
 1162 <SigningTime> output. If there are multiple such values present, behavior is profile-defined.

1163 These options are not allowed in multi-signature verification.

```

1164 <xs:element name="ReturnSigningTime" />

```

1165 The <SigningTime> optional output contains an indication of when the signature was
 1166 performed, and a boolean attribute that indicates whether this value is attested to by a third-party
 1167 timestamp authority (if true), or only by the signer (if false).

```

1168 <xs:element name="SigningTime">
1169   <xs:complexType>
1170     <xs:simpleContent>
1171       <xs:extension base="xs:dateTime">
1172         <xs:attribute name="ThirdPartyTimestamp"
1173           type="xs:boolean" use="required" />
1174       </xs:extension>
1175     </xs:simpleContent>
1176   </xs:complexType>
1177 </xs:element>

```

1178 **4.6.6 Optional Input <ReturnSignerIdentity> and Output**
1179 **<SignerIdentity>**

1180 The presence of the <ReturnSignerIdentity> optional input instructs the server to return a
1181 <SignerIdentity> output.

1182 This optional input and output are not allowed in multi-signature verification.

```
1183 <xs:element name="ReturnSignerIdentity" />
```

1184 The <SignerIdentity> optional output contains an indication of who performed the signature.

```
1185 <xs:element name="SignerIdentity" type="saml:NameIdentifierType" />
```

1186 **4.6.7 Optional Input <ReturnUpdatedSignature> and Output**
1187 **<UpdatedSignature>**

1188 The presence of the <ReturnUpdatedSignature> optional input instructs the server to return
1189 an <UpdatedSignature> output, containing a new or updated signature.

1190 The Type attribute on <ReturnUpdatedSignature>, if present, defines exactly what it means
1191 to "update" a signature. For example, the updated signature may be the original signature with
1192 some additional unsigned signature properties added to it (such as timestamps, counter-
1193 signatures, or additional information for use in verification), or the updated signature could be an
1194 entirely new signature calculated on the same input documents as the input signature. Profiles
1195 that use this optional input MUST define the allowed values and their semantics, and the default
1196 value, for the Type attribute (unless only a single type of updated signature is supported, in which
1197 case the Type attribute can be omitted).

1198 Multiple occurrences of this optional input can be present in a single verify request message. If
1199 multiple occurrences are present, each occurrence MUST have a different Type attribute. Each
1200 occurrence will generate a corresponding optional output. These optional outputs SHALL be
1201 distinguishable based on their Type attribute, which will match each output with an input.

1202 These options are not allowed in multi-signature verification.

```
1203 <xs:element name="ReturnUpdatedSignature">  
1204 <xs:complexType>  
1205 <xs:attribute name="Type" type="xs:anyURI" use="optional" />  
1206 </xs:complexType>  
1207 </xs:element>
```

1208 The <UpdatedSignature> optional output contains the returned signature.

```
1209 <xs:element name="UpdatedSignature">  
1210 <xs:complexType>  
1211 <xs:sequence>  
1212 <xs:element ref="dss:SignatureObject" />  
1213 <xs:sequence>  
1214 <xs:attribute name="Type" type="xs:anyURI" use="optional" />  
1215 </xs:complexType>  
1216 </xs:element>
```

1217
1218

4.6.8 Optional Input <ReturnTransformedDocument> and Output <TransformedDocument>

1219 The <ReturnTransformedDocument> optional input instructs the server to return an input
1220 document to which the XML signature transforms specified by a particular <ds:Reference>
1221 have been applied. The <ds:Reference> is indicated by the zero-based WhichReference
1222 attribute (0 means the first <ds:Reference> in the signature, 1 means the second, and so on).
1223 Multiple occurrences of this optional input can be present in a single verify request message.
1224 Each occurrence will generate a corresponding optional output.
1225 These options are not allowed in multi-signature verification.

1226
1227
1228
1229
1230
1231

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

1232 The <TransformedDocument> optional output contains a document corresponding to the
1233 specified <ds:Reference>, after all the transforms in the reference have been applied. In other
1234 words, the hash value of the returned document should equal the <ds:Reference> element's
1235 <ds:DigestValue>. To match outputs to inputs, each <TransformedDocument> will contain
1236 a WhichReference attribute which matches the corresponding optional input.

1237
1238
1239
1240
1241
1242
1243
1244
1245

```
<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>
```

1246

1247

5 DSS Core Elements

1248

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

1249

1250

5.1 Element <Timestamp>

1251

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].

1252

1253

1254

1255

1256

An XML timestamp may contain:

1257

<ds:Signature> [Optional]

1258

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

1259

<RFC3161TimeStampToken> [Optional]

1260

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

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

```
<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>
```

1271

5.1.1 XML Timestamp Token

1272

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).

1273

1274

1275

1276

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

1277

<ds:KeyInfo> [Required]

1278

1279

1280

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.

1281

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

1282

1283

1284

1285

1286

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.

1287 <ds:Object> [Required]

1288 A <TstInfo> element SHALL be contained in a <ds:Object> element.

1289 **5.1.2 Element <TstInfo>**

1290 A <TstInfo> element is included in an XML timestamp token as a
1291 <ds:Signature>/<ds:Object> child element. A <TstInfo> element has the following
1292 children:

1293 <SerialNumber> [Required]

1294 This element SHALL contain a serial number produced by the timestamp authority (TSA).
1295 It MUST be unique across all the tokens issued by a particular TSA.

1296 <CreationTime> [Required]

1297 The time at which the token was issued.

1298 <Policy> [Optional]

1299 This element SHALL identify the policy under which the token was issued. The TSA's
1300 policy SHOULD identify the fundamental source of its time.

1301 <ErrorBound> [Optional]

1302 The TSA's estimate of the maximum error in its local clock.

1303 <Ordered> [Default="false"]

1304 This element SHALL indicate whether or not timestamps issued by this TSA, under this
1305 policy, are strictly ordered according to the value of the CreationTime element value.

1306 TSA [Optional]

1307 The name of the TSA.

```
1308 <xs:element name="TstInfo">
1309   <xs:complexType>
1310     <xs:sequence>
1311       <xs:element name="SerialNumber" type="xs:integer"/>
1312       <xs:element name="CreationTime" type="xs:dateTime"/>
1313       <xs:element name="Policy" type="xs:anyURI" minOccurs="0"/>
1314       <xs:element name="ErrorBound" type="xs:duration"
1315         minOccurs="0"/>
1316       <xs:element name="Ordered" type="xs:boolean"
1317         default="false" minOccurs="0"/>
1318       <xs:element name="TSA" type="saml:NameIdentifierType"
1319         minOccurs="0"/>
1320     </xs:sequence>
1321   </xs:complexType>
1322 </xs:element>
```

1323 **5.1.3 Timestamp verification procedure**

1324 If any one of these steps results in failure, then the timestamp token SHOULD be rejected.

- 1325 1. Locate and verify the signature-verification key corresponding to the ds:KeyInfo/
1326 element contents.
- 1327 2. Verify that the signature-verification key is authorized for verifying timestamps.

- 1328 3. Verify that the signature-verification key conforms with all relevant aspects of the relying-
1329 party's policy.
- 1330 4. Verify that all digest and signature algorithms conform with the relying-party's policy.
- 1331 5. Verify that the signature-verification key is consistent with the
1332 ds:SignedInfo/SignatureMethod/@Algorithm element value.
- 1333 6. Verify that there is a single ds:SignedInfo/Reference element whose URI attribute
1334 references a <ds:Object> containing an enveloped <TstInfo> element.
- 1335 7. Verify that there is a ds:SignedInfo/Reference element whose URI attribute
1336 correctly identifies the timestamped document.
- 1337 8. Verify that the tstInfo/Policy element value is acceptable.
- 1338 9. Verify all digests and the signature.
- 1339 If comparing the tstInfo/CreationTime element value to another time value, first verify that
1340 they differ by more than the error bound value.

1341 5.2 Element <RequesterIdentity>

1342 This section contains the definition of an XML Requester Identity element. This element can be
1343 used as a signature property in an XML signature to identify the client who requested the
1344 signature.

1345 This element has the following children:

1346 Name [Required]

1347 The name or role of the requester who requested the signature be performed.

1348 SupportingInfo [Optional]

1349 Information supporting the name (such as a SAML Assertion [SAMLCore1.1], Liberty Alliance
1350 Authentication Context, or X.509 Certificate).

1351 The following schema fragment defines the <RequesterIdentity> element:

```
1352 <xs:element name="RequesterIdentity">  
1353   <xs:complexType>  
1354     <xs:sequence>  
1355       <xs:element name="Name" type="saml:NameIdentifierType"/>  
1356       <xs:element name="SupportingInfo" type="dss:AnyType"  
1357         minOccurs="0"/>  
1358     </xs:sequence>  
1359   </xs:complexType>  
1360 </xs:element>
```

1361

6 DSS Core Bindings

1362 Mappings from DSS messages into standard communications protocols are called DSS *bindings*.
1363 *Transport bindings* specify how DSS messages are encoded and carried over some lower-level
1364 transport protocol. *Security bindings* specify how confidentiality, authentication, and integrity can
1365 be achieved for DSS messages in the context of some transport binding.

1366 Below we specify an initial set of bindings for DSS. Future bindings may be introduced by the
1367 OASIS DSS TC or by other parties.

1368

6.1 HTTP POST Transport Binding

1369 In this binding, the DSS request/response exchange occurs within an HTTP POST exchange
1370 [RFC 2616]. The following rules apply to the HTTP request:

- 1371 1. The client may send an HTTP/1.0 or HTTP/1.1 request.
- 1372 2. The Request URI may be used to indicate a particular service endpoint.
- 1373 3. The `Content-Type` header MUST be set to "application/xml".
- 1374 4. The `Content-Length` header MUST be present and correct.
- 1375 5. The DSS request message MUST be sent in the body of the HTTP Request.

1376 The following rules apply to the HTTP Response:

- 1377 3. The `Content-Type` header MUST be set to "text/xml".
- 1378 4. The `Content-Length` header MUST be present and correct.
- 1379 5. The DSS response message MUST be sent in the body of the HTTP Response.
- 1380 6. The HTTP status code MUST be set to 200 if a DSS response message is returned.
1381 Otherwise, the status code can be set to 3xx to indicate a redirection, 4xx to indicate a
1382 low-level client error (such as a malformed request), or 5xx to indicate a low-level server
1383 error.

1384

6.2 SOAP 1.2 Transport Binding

1385 In this binding, the DSS request/response exchange occurs using the SOAP 1.2 message
1386 protocol [SOAP]. The following rules apply to the SOAP request:

- 1387 1. A single DSS `<SignRequest>` or `<VerifyRequest>` element will be transmitted within
1388 the body of the SOAP message.
- 1389 2. The client MUST NOT include any additional XML elements in the SOAP body.
- 1390 3. The UTF-8 character encoding must be used for the SOAP message.
- 1391 4. Arbitrary SOAP headers may be present.

1392 The following rules apply to the SOAP response:

- 1393 1. The server MUST return either a single DSS `<SignResponse>` or `<VerifyResponse>`
1394 element within the body of the SOAP message, or a SOAP fault code.
- 1395 2. The server MUST NOT include any additional XML elements in the SOAP body.
- 1396 3. If a DSS server cannot parse a DSS request, or there is some error with the SOAP
1397 envelope, the server MUST return a SOAP fault code. Otherwise, a DSS result code
1398 should be used to signal errors.

- 1399 4. The UTF-8 character encoding must be used for the SOAP message.
1400 5. Arbitrary SOAP headers may be present.
1401 6. On receiving a DSS response in a SOAP message, the client MUST NOT send a fault
1402 code to the DSS server.

1403 6.3 TLS Security Bindings

1404 TLS [RFC 2246] is a session-security protocol that can provide confidentiality, authentication, and
1405 integrity to the HTTP POST transport binding, the SOAP 1.2 transport binding, or others. TLS
1406 supports a variety of authentication methods, so we define several security bindings below. All of
1407 these bindings inherit the following rules:

- 1408 1. TLS 1.0 MUST be supported. SSL 3.0 MAY be supported. Future versions of TLS MAY
1409 be supported.
1410 2. RSA ciphersuites MUST be supported. Diffie-Hellman and DSS ciphersuites MAY be
1411 supported.
1412 3. TripleDES ciphersuites MUST be supported. AES ciphersuites SHOULD be supported.
1413 Other ciphersuites MAY be supported, except for weak ciphersuites intended to meet
1414 export restrictions, which SHOULD NOT be supported.

1415 6.3.1 TLS X.509 Server Authentication

1416 The following ciphersuites defined in [RFC 2246] and [RFC 3268] are supported. The server
1417 MUST authenticate itself with an X.509 certificate chain [RFC 3280]. The server MUST NOT
1418 request client authentication.

1419 **MUST:**

1420 TLS_RSA_WITH_3DES_EDE_CBC_SHA

1421 **SHOULD:**

1422 TLS_RSA_WITH_AES_128_CBC_SHA

1423 TLS_RSA_WITH_AES_256_CBC_SHA

1424 6.3.2 TLS X.509 Mutual Authentication

1425 The same ciphersuites mentioned in section 6.2.1 are supported. The server MUST authenticate
1426 itself with an X.509 certificate chain, and MUST request client authentication. The client MUST
1427 authenticate itself with an X.509 certificate chain.

1428 6.3.3 TLS SRP Authentication

1429 SRP is a way of using a username and password to accomplish mutual authentication. The
1430 following ciphersuites defined in [draft-ietf-tls-srp-08] are supported.

1431 **MUST:**

1432 TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA

1433 **SHOULD:**

1434 TLS_SRP_SHA_WITH_AES_128_CBC_SHA

1435 TLS_SRP_SHA_WITH_AES_256_CBC_SHA

1436 **6.3.4 TLS SRP and X.509 Server Authentication**

1437 SRP can be combined with X.509 server authentication. The following ciphersuites defined in
1438 **[draft-ietf-tls-srp-08]** are supported.

1439 **MUST:**

1440 TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA

1441 **SHOULD:**

1442 TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA

1443 TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA

1444 7 DSS-Defined Identifiers

1445 The following sections define various URI-based identifiers. Where possible an existing URN is
1446 used to specify a protocol. In the case of IETF protocols the URN of the most current RFC that
1447 specifies the protocol is used (see [RFC 2648]). URI references created specifically for DSS
1448 have the following stem:

1449 urn:oasis:names:tc:dss:1.0:

1450 7.1 Signature Type Identifiers

1451 The following identifiers MAY be used as the content of the <SignatureType> optional input
1452 (see section 3.5.1).

1453 7.1.1 XML Signature

1454 **URI:** urn:ietf:rfc:3275

1455 This refers to an XML signature per [XMLSig].

1456 7.1.2 XML TimeStampToken

1457 **URI:** urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken

1458 This refers to an XML timestamp containing an XML signature, per section 5.1.

1459 7.1.3 RFC 3161 TimeStampToken

1460 **URI:** urn:ietf:rfc:3161

1461 This refers to an XML timestamp containing an ASN.1 TimeStampToken, per [RFC 3161].

1462 7.1.4 CMS Signature

1463 **URI:** urn:ietf:rfc:3369

1464 This refers to a CMS signature per [RFC 3369].

1465 7.1.5 PGP Signature

1466 **URI:** urn:ietf:rfc:2440

1467 This refers to a PGP signature per [RFC 2440].

1468

8 Editorial Issues

1469

1470

1471

1472

1473

1474

- 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.

1475

1476

- 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.

1477

1478

- 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.

1479

1480

1481

1482

- 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.

1483

1484

1485

- 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.

1486

1487

1488

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

Resolution: Yes - `<OptionalInputs>` and `<OptionalOutputs>`

1489

1490

- 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

1491

1492

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

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

1493

1494

1495

1496

1497

- 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.

1498

1499

1500

1501

1502

1503

1504

1505

- 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.

1506

1507

- 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.

1508

1509

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

Resolution: URL (in draft 17)

1510

- 1511 13) Should we add a WSS Security Binding?
1512 **Resolution:** not now
1513 14) Should we add some way for an external policy authority to vouch for some portion of a
1514 request?
1515 **Resolution:** not in the core
1516 15) Should RequestID be removed?
1517 **Resolution:** No.
1518 16) Should input documents have a RefId attribute?
1519 **Resolution:** No.
1520 17) Should <SignaturePtr> be optional when there's only 1 input doc, with 1 signature?
1521 **Resolution:** Yes.
1522 18) Should the server return the <Profile> it used?
1523 **Resolution:** Yes.
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546

1547

1548

9 References

1549

9.1 Normative

- 1550 **[Core-XSD]** T. Perrin et al. *DSS Schema*. OASIS, **(MONTH/YEAR TBD)**
- 1551 **[RFC 2119]** S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*.
1552 IETF RFC 2396, August 1998.
1553 <http://www.ietf.org/rfc/rfc2396.txt>.
- 1554 **[RFC 2246]** T. Dierks, C. Allen. *The TLS Protocol Version 1.0*. IETF RFC 2246,
1555 January 1999.
1556 <http://www.ietf.org/rfc/rfc2246.txt>.
- 1557 **[RFC 2396]** T. Berners-Lee et al. *Uniform Resource Identifiers (URI): Generic
1558 Syntax*. IETF RFC 2396, August 1998.
1559 <http://www.ietf.org/rfc/rfc2396.txt>.
- 1560 **[RFC 2440]** J. Callas, L. Donnerhacke, H. Finney, R. Thayer. *OpenPGP Message
1561 Format*. IETF RFC 2440, November 1998.
1562 <http://www.ietf.org/rfc/rfc2440.txt>.
- 1563 **[RFC 2616]** R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC
1564 2616, June 1999.
1565 <http://www.ietf.org/rfc/rfc2616.txt>.
- 1566 **[RFC 2648]** R. Moats. *A URN Namespace for IETF Documents*. IETF RFC 2648,
1567 August 1999.
1568 <http://www.ietf.org/rfc/rfc2648.txt>.
- 1569 **[RFC 2822]** P. Resnick. *Internet Message Format*. IETF RFC 2822, April 2001.
1570 <http://www.ietf.org/rfc/rfc2822.txt>
- 1571 **[RFC 3161]** C. Adams, P. Cain, D. Pinkas, R. Zuccherato. *Internet X.509 Public Key
1572 Infrastructure Time-Stamp Protocol (TSP)*. IETF RFC 3161, August
1573 2001.
1574 <http://www.ietf.org/rfc/rfc3161.txt>.
- 1575 **[RFC 3268]** P. Chown. *AES Ciphersuites for TLS*. IETF RFC 3268, June 2002.
1576 <http://www.ietf.org/rfc/rfc3268.txt>.
- 1577 **[RFC 3280]** R. Housley, W. Polk, W. Ford, D. Solo. *Internet X.509 Public Key
1578 Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.
1579 IETF RFC 3280, April 2002.
1580 <http://www.ietf.org/rfc/rfc3280.txt>.
- 1581 **[RFC 3369]** R. Housley. *Cryptographic Message Syntax*. IETF RFC 3369, August
1582 2002.
1583 <http://www.ietf.org/rfc/rfc2459.txt>.
- 1584 **[SAMLCore1.1]** E. Maler et al. *Assertions and Protocol for the OASIS Security Assertion
1585 Markup Language (SAML) V 1.1*. OASIS, November 2002.
1586 [http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-
1587 saml-core-1.1.pdf](http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf)
- 1588 **[Schema1]** H. S. Thompson et al. *XML Schema Part 1: Structures*. W3C
1589 Recommendation, May 2001.
1590 <http://www.w3.org/TR/xmlschema-1/>
- 1591 **[SOAP]** M. Gudgin et al. *SOAP Version 1.2 Part 1: Messaging Framework*.
1592 W3C Recommendation, June 2003.
1593 <http://www.w3.org/TR/xmlschema-1/>
- 1594 **[XML-C14N]** J. Boyer. *Canonical XML Version 1.0*. W3C Recommendation, March
1595 2001.
1596 <http://www.w3.org/TR/xml-c14n>

1597	[XML-ns]	T. Bray, D. Hollander, A. Layman. <i>Namespaces in XML</i> . W3C Recommendation, January 1999.
1598		
1599		http://www.w3.org/TR/1999/REC-xml-names-19990114
1600	[XMLSig]	D. Eastlake et al. <i>XML-Signature Syntax and Processing</i> . W3C Recommendation, February 2002.
1601		
1602		http://www.w3.org/TR/1999/REC-xml-names-19990114
1603	[XML-TSP]	T. Perrin et al. <i>XML Timestamping Profile of the OASIS Digital Signature Services</i> . W3C Recommendation, February 2002. OASIS,
1604		(MONTH/YEAR TBD)
1605		
1606		
1607		
1608		
1609		
1610		
1611		
1612		
1613		

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>

1615

Appendix B. Notices

1616 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
1617 that might be claimed to pertain to the implementation or use of the technology described in this
1618 document or the extent to which any license under such rights might or might not be available;
1619 neither does it represent that it has made any effort to identify any such rights. Information on
1620 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
1621 website. Copies of claims of rights made available for publication and any assurances of licenses
1622 to be made available, or the result of an attempt made to obtain a general license or permission
1623 for the use of such proprietary rights by implementors or users of this specification, can be
1624 obtained from the OASIS Executive Director.

1625 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
1626 applications, or other proprietary rights which may cover technology that may be required to
1627 implement this specification. Please address the information to the OASIS Executive Director.

1628 Copyright © OASIS Open 2003. *All Rights Reserved.*

1629 This document and translations of it may be copied and furnished to others, and derivative works
1630 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
1631 published and distributed, in whole or in part, without restriction of any kind, provided that the
1632 above copyright notice and this paragraph are included on all such copies and derivative works.
1633 However, this document itself does not be modified in any way, such as by removing the
1634 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
1635 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
1636 Property Rights document must be followed, or as required to translate it into languages other
1637 than English.

1638 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
1639 successors or assigns.

1640 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1641 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
1642 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
1643 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
1644 PARTICULAR PURPOSE.