



Digital Signature Service Core Protocols, Elements, and Bindings Version 1.0

OASIS Standard

11 April 2007

Specification URIs:

This Version:

<http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.html>
<http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf>
<http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.doc>

Latest Version:

<http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.html>
<http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf>
<http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.doc>

Technical Committee:

OASIS Digital Signature Services TC

Chair(s):

Nick Pope, Thales eSecurity
Juan Carlos Cruellas, Centre d'aplicacions avançades d'Internet (UPC)

Editor(s):

Stefan Drees, *individual* <stefan@zinz.name>

Related work:

Declared XML Namespace(s):

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

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 document was last revised or approved by the membership of OASIS on the above date. The level of approval is also listed above. Check the current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/dss/>.

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 Technical Committee web page (<http://www.oasis-open.org/committees/dss/ipr.php>).

The non-normative errata page for this specification is located at <http://www.oasis-open.org/committees/dss/>.

Notices

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

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

Copyright © OASIS® 1993–2007. All Rights Reserved.

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

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

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

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

Table of Contents

1	Introduction.....	7
1.1	Terminology.....	7
1.2	Normative References.....	7
1.3	Schema Organization and Namespaces.....	9
1.4	DSS Overview (Non-normative).....	9
2	Common Protocol Structures.....	11
2.1	Type AnyType.....	11
2.2	Type InternationalStringType.....	11
2.3	Type saml:NameIdentifierType.....	11
2.4	Element <InputDocuments>.....	11
2.4.1	Type DocumentBaseType.....	12
2.4.2	Element <Document>.....	13
2.4.3	Element <TransformedData>.....	14
2.4.4	Element <DocumentHash>.....	15
2.5	Element <SignatureObject>.....	16
2.6	Element <Result>.....	17
2.7	Elements <OptionalInputs> and <OptionalOutputs>.....	19
2.8	Common Optional Inputs.....	20
2.8.1	Optional Input <ServicePolicy>.....	20
2.8.2	Optional Input <ClaimedIdentity>.....	20
2.8.3	Optional Input <Language>.....	20
2.8.4	Optional Input <AdditionalProfile>.....	21
2.8.5	Optional Input <Schemas>.....	21
2.9	Common Optional Outputs.....	21
2.9.1	Optional Output <Schemas>.....	21
2.10	Type <RequestBaseType>.....	22
2.11	Type <ResponseBaseType>.....	22
2.12	Element <Response>.....	23
3	The DSS Signing Protocol.....	24
3.1	Element <SignRequest>.....	24
3.2	Element <SignResponse>.....	24
3.3	Processing for XML Signatures.....	25
3.3.1	Basic Process for <Base64XML>.....	25
3.3.2	Process Variant for <InlineXML>.....	26
3.3.3	Process Variant for <EscapedXML>.....	26
3.3.4	Process Variant for <Base64Data>.....	26
3.3.5	Process Variant for <TransformedData>.....	27
3.3.6	Process Variant for <DocumentHash>.....	27
3.4	Basic Processing for CMS Signatures.....	28
3.4.1	Process Variant for <DocumentHash>.....	28
3.5	Optional Inputs and Outputs.....	29
3.5.1	Optional Input <SignatureType>.....	29
3.5.2	Optional Input <AddTimestamp>.....	29

3.5.3	Optional Input <IntendedAudience>.....	31
3.5.4	Optional Input <KeySelector>	31
3.5.5	Optional Input <Properties>	31
3.5.6	Optional Input <IncludeObject>.....	32
3.5.7	Optional Input <IncludeEContent>	34
3.5.8	Enveloped Signatures, Optional Input <SignaturePlacement> and Output <DocumentWithSignature>	34
3.5.9	Optional Input <SignedReferences>	36
4	The DSS Verifying Protocol.....	38
4.1	Element <VerifyRequest>.....	38
4.2	Element <VerifyResponse>	38
4.3	Basic Processing for XML Signatures	38
4.3.1	Multi-Signature Verification	40
4.3.2	Signature Timestamp verification procedure.....	40
4.4	Basic Processing for CMS Signatures.....	42
4.5	Optional Inputs and Outputs.....	42
4.5.1	Optional Input <VerifyManifests> and Output <VerifyManifestResults>	42
4.5.2	Optional Input <UseVerificationTime>	43
4.5.3	Optional Input/Output <ReturnVerificationTimeInfo> / <VerificationTimeInfo>	44
4.5.4	Optional Input <AdditionalKeyInfo>.....	45
4.5.5	Optional Input <ReturnProcessingDetails> and Output <ProcessingDetails>.....	45
4.5.6	Optional Input <ReturnSigningTimeInfo> and Output <SigningTimeInfo>	46
4.5.7	Optional Input <ReturnSignerIdentity> and Output <SignerIdentity>.....	47
4.5.8	Optional Input <ReturnUpdatedSignature> and Outputs <DocumentWithSignature>, <UpdatedSignature>	47
4.5.9	Optional Input <ReturnTransformedDocument> and Output <TransformedDocument>.....	49
4.5.10	Optional Input <ReturnTimestampedSignature> and Outputs <DocumentWithSignature>, <TimestampedSignature>	49
5	DSS Core Elements	51
5.1	Element <Timestamp>.....	51
5.1.1	XML Timestamp Token	51
5.1.2	Element <TstInfo>	52
5.2	Element <RequesterIdentity>	52
6	DSS Core Bindings.....	54
6.1	HTTP POST Transport Binding	54
6.2	SOAP 1.2 Transport Binding	54
6.2.1	SOAP Attachment Feature and Element <AttachmentReference>	55
6.3	TLS Security Bindings	56
6.3.1	TLS X.509 Server Authentication	57
6.3.2	TLS X.509 Mutual Authentication.....	57
6.3.3	TLS SRP Authentication.....	57
6.3.4	TLS SRP and X.509 Server Authentication	57
7	DSS-Defined Identifiers.....	58
7.1	Signature Type Identifiers.....	58
7.1.1	XML Signature.....	58
7.1.2	XML TimeStampToken.....	58

7.1.3 RFC 3161 TimeStampToken.....	58
7.1.4 CMS Signature	58
7.1.5 PGP Signature.....	58
A. Use of Exclusive Canonicalization	59
B. More Complex <Response> Example.....	60
C. Acknowledgements	61

1 Introduction

[All text is normative unless otherwise labeled]

1.1 Terminology

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 Normative References

- [Core-XSD] S. Drees,. *DSS Schema*. OASIS, February 2007.
- [DSS-TS-P] T Perrin et al. *DSS Timestamp Profile*. OASIS, February 2007.
- [DSS-AdES-P] JC Cruellas et al. *Advanced Electronic Signature Profiles of the OASIS Digital Signature Service*. OASIS, February 2007
- [RFC 2119] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. IETF RFC 2119, March 1997.
<http://www.ietf.org/rfc/rfc2119.txt>.
- [RFC 2246] T Dierks, C. Allen. *The TLS Protocol Version 1.0*. IETF RFC 2246, January 1999.
<http://www.ietf.org/rfc/rfc2246.txt>.
- [RFC 2396] T. Berners-Lee et al. *Uniform Resource Identifiers (URI): Generic Syntax*. IETF RFC 2396, August 1998.
<http://www.ietf.org/rfc/rfc2396.txt>.
- [RFC 2440] J. Callas, L. Donnerhackle, H. Finney, R. Thayer. *OpenPGP Message Format*. IETF RFC 2440, November 1998.
<http://www.ietf.org/rfc/rfc2440.txt>.
- [RFC 2616] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC 2616, June 1999.
<http://www.ietf.org/rfc/rfc2616.txt>.
- [RFC 2648] R. Moats. *A URN Namespace for IETF Documents*. IETF RFC 2648, August 1999.
<http://www.ietf.org/rfc/rfc2648.txt>.
- [RFC 2822] P. Resnick. *Internet Message Format*. IETF RFC 2822, April 2001.
<http://www.ietf.org/rfc/rfc2822.txt>
- [RFC 3161] C. Adams, P. Cain, D. Pinkas, R. Zuccherato. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. IETF RFC 3161, August 2001.
<http://www.ietf.org/rfc/rfc3161.txt>.
- [RFC 3268] P. Chown. *AES Ciphersuites for TLS*. IETF RFC 3268, June 2002.
<http://www.ietf.org/rfc/rfc3268.txt>.
- [RFC 3280] R. Housley, W. Polk, W. Ford, D. Solo. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. IETF RFC 3280, April 2002.
<http://www.ietf.org/rfc/rfc3280.txt>.

- 47 **[RFC 3852]** R. Housley. *Cryptographic Message Syntax*. IETF RFC 3852, July 2004.
48 <http://www.ietf.org/rfc/rfc3852.txt>.
49 (Remark: As used in DSS, all implementations based upon RFC 3852, RFC 3369
50 and previous releases of CMS will suffice. For the sake of simplicity the
51 "urn:ietf::3369" is used throughout the document to indicate a CMS message as
52 specified in RFC 3852 or RFC 3369 or any version (including PKCS #7).
- 53 **[SAMLCore1.1]** E. Maler et al. *Assertions and Protocol for the OASIS Security Assertion Markup
54 Language (SAML) V 1.1*. OASIS, November 2002.
55 [http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-
56 1.1.pdf](http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf)
- 57 **[Schema1]** H. S. Thompson et al. *XML Schema Part 1: Structures*. W3C Recommendation,
58 May 2001.
59 <http://www.w3.org/TR/xmlschema-1/>
- 60 **[SOAP]** M. Gudgin et al. *SOAP Version 1.2 Part 1: Messaging Framework*. W3C
61 Recommendation, June 2003.
62 <http://www.w3.org/TR/xmlschema-1/>
- 63 **[SOAPAtt]** H. F. Nielsen, H. Ruellan *SOAP 1.2 Attachment Feature*, W3C Working Group
64 Note, 8 June 2004
65 <http://www.w3.org/TR/soap12-af/>
- 66 **[WS-I-Att]** Ch. Ferris, A. Karmarkar, C. K. Liu *Attachments Profile Version 1.0*, The Web
67 Services-Interoperability Organization (WS-I), 20 April 2006
68 <http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html>
- 69 **[XML-C14N]** J. Boyer. *Canonical XML Version 1.0*. W3C Recommendation, March 2001.
70 <http://www.w3.org/TR/xml-c14n>
- 71 **[XML-ESCAPE]** Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, et al. *Predefined Entities in
72 Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C Recommendation,
73 04 February 2004,
74 <http://www.w3.org/TR/REC-xml/#dt-escape>
- 75 **[xml:id]** xml:id, Version 1.0, W3C Recommendation, 9 September 2005,
76 <http://www.w3.org/TR/xml-id/>
- 77 **[XML-ns]** T. Bray, D. Hollander, A. Layman. *Namespaces in XML*. W3C
78 Recommendation, January 1999.
79 <http://www.w3.org/TR/1999/REC-xml-names-19990114>
- 80 **[XML-NT-Document]** <http://www.w3.org/TR/2004/REC-xml-20040204/#NT-document>
- 81 **[XML-PROLOG]** Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, et al. *Prolog and Document
82 Type Declaration in Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C
83 Recommendation, 04 February 2004, [http://www.w3.org/TR/REC-xml/#sec-
84 prolog-dtd](http://www.w3.org/TR/REC-xml/#sec-prolog-dtd)
- 85 **[XMLDSIG]** D. Eastlake et al. *XML-Signature Syntax and Processing*. W3C
86 Recommendation, February 2002.
87 <http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>
- 88 **[XML-TSP]** T. Perrin et al. *XML Timestamping Profile of the OASIS Digital Signature
89 Services*. W3C Recommendation, February 2002. OASIS, **(MONTH/YEAR
90 TBD)**
- 91 **[XML]** Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation 04 February
92 2004 <http://www.w3.org/TR/REC-xml/#sec-element-content>
- 93 **[XPath]** XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999
94 <http://www.w3.org/TR/xpath>
- 95 **[XML-xcl-c14n]** Exclusive XML Canonicalization Version 1.0. W3C Recommendation 18 July 2002
96 <http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/>

97 1.3 Schema Organization and Namespaces

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

101 This schema is associated with the following XML namespace:

```
102 urn:oasis:names:tc:dss:1.0:core:schema
```

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

104 Conventional XML namespace prefixes are used in the schema:

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

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

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

```
114 <xs:schema xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema "  
115           xmlns:ds="http://www.w3.org/2000/09/xmlsig#"  
116           xmlns:xs="http://www.w3.org/2001/XMLSchema "  
117           xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"  
118           targetNamespace="urn:oasis:names:tc:dss:1.0:core:schema "  
119           elementFormDefault="qualified"  
120           attributeFormDefault="unqualified">  
121   <xs:annotation>  
122     <xs:documentation xml:lang="en">This Schema defines the Digital Signature  
123     Service Core Protocols, Elements, and Bindings Committee Draft 5 for Public  
124     Review</xs:documentation>  
125   </xs:annotation>  
126   <xs:import namespace="http://www.w3.org/2000/09/xmlsig#"  
127     schemaLocation="http://www.w3.org/TR/xmlsig-core/xmlsig-core-schema.xsd"/>  
128   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:assertion"  
129     schemaLocation="http://www.oasis-open.org/committees/download.php/3408/oasis-  
130     sstc-saml-schema-protocol-1.1.xsd"/>  
131   <xs:import namespace="http://www.w3.org/XML/1998/namespace"  
132     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
```

133 1.4 DSS Overview (Non-normative)

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

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

142 The signing and verifying protocols are chiefly designed to support the creation and verification of XML
143 signatures **[XMLDSIG]**, XML timestamps (see section 5.1), binary timestamps **[RFC 3161]** and CMS
144 signatures **[RFC 3852]**. These protocols may also be extensible to other types of signatures and
145 timestamps, such as PGP signatures **[RFC 2440]**.

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

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

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

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

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

173

174

175 2 Common Protocol Structures

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

177 2.1 Type AnyType

178 The **AnyType** complex type allows arbitrary XML element content within an element of this type (see
179 section 3.2.1 Element Content [XML]).

```
180 <xs:complexType name="AnyType">  
181   <xs:sequence>  
182     <xs:any processContents="lax"  
183       minOccurs="0"  
184       maxOccurs="unbounded"/>  
185   </xs:sequence>  
186 </xs:complexType>
```

187 2.2 Type InternationalStringType

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

```
190 <xs:complexType name="InternationalStringType">  
191   <xs:simpleContent>  
192     <xs:extension base="xs:string">  
193       <xs:attribute ref="xml:lang" use="required"/>  
194     </xs:extension>  
195   </xs:simpleContent>  
196 </xs:complexType>
```

197 2.3 Type saml:NameIdentifierType

198 The **saml:NameIdentifierType** complex type is used where different types of names are needed (such
199 as email addresses, Distinguished Names, etc.). This type is borrowed from [SAMLCore1.1] section
200 2.4.2.2. It consists of a string with the following attributes:

201 NameQualifier [Optional]

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

204 Format [Optional]

205 A URI [RFC 2396] reference representing the format in which the string is provided. See section 7.3
206 of [SAMLCore1.1] for some URI references that may be used as the value of the `Format` attribute.

207 2.4 Element <InputDocuments>

208 The `<InputDocuments>` element is used to send input documents to a DSS server, whether for signing
209 or verifying. An input document can be any piece of data that can be used as input to a signature or
210 timestamp calculation. An input document can even *be* a signature or timestamp (for example, a pre-
211 existing signature can be counter-signed or timestamped). An input document could also be a
212 `<ds:Manifest>`, allowing the client to handle manifest creation while using the server to create the rest
213 of the signature. Manifest validation is supported by an optional input / output.

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

215 `<Document>` [Any Number]

216 It contains a document as specified in section 2.4.2 of this document.

217 <TransformedData> [Any Number]
218 This contains the binary output of a chain of transforms applied by a client as specified in section 2.4.3
219 of this document.
220 <DocumentHash> [Any Number]
221 This contains the hash value of an XML document or some other data after a client has applied a
222 sequence of transforms and also computed a hash value as specified in section 2.4.4 of this
223 document.
224 <Other>
225 Other may contain arbitrary content that may be specified in a profile and can also be used to extend
226 the Protocol for details see section 2.1.

```
227 <xs:element name="InputDocuments">  
228 <xs:complexType>  
229 <xs:sequence>  
230 <xs:choice minOccurs="1" maxOccurs="unbounded">  
231 <xs:element ref="dss:Document"/>  
232 <xs:element ref="dss:TransformedData"/>  
233 <xs:element ref="dss:DocumentHash"/>  
234 <xs:element name="Other" type="dss:AnyType"/>  
235 </xs:choice>  
236 </xs:sequence>  
237 </xs:complexType>  
238 </xs:element>
```

239 When using DSS to create or verify XML signatures, each input document will usually correspond to a
240 single <ds:Reference> element. Thus, in the descriptions below of the <Document>,
241 <TransformedData> and <DocumentHash> elements, it is explained how certain elements and
242 attributes of a <Document>, <TransformedData> and <DocumentHash> correspond to components
243 of a <ds:Reference>.

244 2.4.1 Type DocumentBaseType

245 The **DocumentBaseType** complex type is subclassed by <Document>, <TransformedData> and
246 <DocumentHash> elements. It contains the basic information shared by subclasses and remaining
247 persistent during the process from input document retrieval until digest calculation for the relevant
248 document. It contains the following elements and attributes:

249 ID [Optional]

250 This identifier gives the input document a unique label within a particular request message. Through
251 this identifier, an optional input (see sections 2.7, 3.5.6 and 3.5.8) can refer to a particular input
252 document.

253 RefURI [Optional]

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

257 RefType [Optional]

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

260 SchemaRefs [Optional]:

261 The identified schemas are to be used to identify ID attributes during parsing in sections 2.5.2, 3.3.1
262 1.a and 4.3 and for XPath evaluation in sections 2.6, 3.5.7, 4.3.1. If anything else but <Schema> are
263 referred to, the server MUST report an error. If a referred to <Schema> is not used by the XML
264 document instance this MAY be ignored or reported to the client in the <Result>/<ResultMessage>
265 (for the definition of <Schema> see 2.8.5 or 2.9.1 on <Schemas>).

266 The Document is assumed to be valid against the first <Schema> referred to by SchemaRefs.
267 If a <Schemas> element is referred to first by SchemaRefs the document is assumed to be valid
268 against the first <Schema> inside <Schemas>. In both cases, the remaining schemas may occur in
269 any order and are used either directly or indirectly by the first schema.
270 If present, the server MUST use the schemas to identify the ID attributes and MAY also perform
271 complete validation against the schemas.

```
272 <xs:complexType name="DocumentBaseType" abstract="true">  
273   <xs:attribute name="ID" type="xs:ID" use="optional"/>  
274   <xs:attribute name="RefURI" type="xs:anyURI" use="optional"/>  
275   <xs:attribute name="RefType" type="xs:anyURI" use="optional"/>  
276   <xs:attribute name="SchemaRefs" type="xs:IDREFS" use="optional"/>  
277 </xs:complexType>
```

278 Note: It is recommended to use `xml:id` as defined in **[xml:id]** as id in the payload being referenced by a
279 <ds:Reference>, because the schema then does not have to be supplied for identifying the ID
280 attributes.

281 2.4.2 Element <Document>

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

284 If the content inside one of the following mutually exclusive elements <InlineXML>, <EscapedXML>
285 or <Base64XML> is not parseable XML data, after appropriate decoding, then the server MUST return a
286 <Result> (section 2.6) issuing a <ResultMajor> RequesterError qualified by a <ResultMinor>
287 NotParseableXMLDocument.

288 The server MUST use the <Schema> referred by <SchemaRefs> for validation if specified.

289 <Base64XML> [Optional] [Default]

290 This contains a base64 string obtained after base64 encoding of a XML data. The server MUST
291 decode it to obtain the XML data.

292 <InlineXML> [Optional]

293 The InlineXMLType clearly expresses the fact, that content of <InlineXML> is inline XML that should
294 be equivalent to a complete XML Document. I.e. having only one DocumentElement (see section 2.1
295 Well-Formed XML Documents **[XML]**) and not allowing anything but PI's and Comments before and
296 after this one element.

297 It may contain the ignorePIs and ignoreComments attributes. These attributes apply to the
298 complete document and indicate respectively, if processing instructions or comments MAY be ignored.

299 If one or both of these attributes are not present, their values MUST be considered to be "true".

300 InlineXML will work with PIs and/or Comments if ignorePIs and ignoreComments are false
301 respectively and if the server supports such behavior.

302 <EscapedXML> [Optional]

303 This contains an escaped string. The server MUST unescape (escape sequences are processed to
304 produce original XML sequence) it for obtaining XML data.

305 <Base64Data> [Optional]

306 This contains a base64 encoding of data that are not XML. The type of data is specified by its
307 MIMEType attribute, that may be required when using DSS with other signature types.

308 <AttachmentReference> [Optional]

309 This contains a reference to an attachment like SOAP attachments or similar data containers that may
310 be passed along with the request. For details see section 6.2.1

```
311 <xs:element name="Document" type="dss:DocumentType"/>  
312
```

```

313 <xs:complexType name="DocumentType">
314   <xs:complexContent>
315     <xs:extension base="dss:DocumentBaseType">
316       <xs:choice>
317         <xs:element name="InlineXML" type="dss:InlineXMLType"/>
318         <xs:element name="Base64XML" type="xs:base64Binary"/>
319         <xs:element name="EscapedXML" type="xs:string"/>
320         <xs:element ref="dss:Base64Data"/>
321         <xs:element ref="dss:AttachmentReference"/>
322       </xs:choice>
323     </xs:extension>
324   </xs:complexContent>
325 </xs:complexType>
326
327 <xs:element name="Base64Data">
328   <xs:complexType>
329     <xs:simpleContent>
330       <xs:extension base="xs:base64Binary">
331         <xs:attribute name="MimeType" type="xs:string"
332           use="optional"/>
333       </xs:extension>
334     </xs:simpleContent>
335   </xs:complexType>
336 </xs:element>
337
338 <xs:complexType name="InlineXMLType">
339   <xs:sequence>
340     <xs:any processContents="lax"/>
341   </xs:sequence>
342   <xs:attribute name="ignorePIs" type="xs:boolean"
343     use="optional" default="true"/>
344   <xs:attribute name="ignoreComments" type="xs:boolean"
345     use="optional" default="true"/>
346 </xs:complexType>

```

347 2.4.3 Element <TransformedData>

348 The <TransformedData> element contains the following elements (in addition to the common ones
349 listed in section 2.4.1):

350 <ds:Transforms> [Required on a SignRequest] [Optional on VerifyRequest]

351 This is the sequence of transforms applied by the client and specifies the value for a
352 <ds:Reference> element's <ds:Transforms> child element. In other words, this specifies
353 transforms that the client has already applied to the input document before the server will hash it.

354 <Base64Data> [Required]

355 This gives the binary output of a sequence of transforms to be hashed at the server side.

356 WhichReference [Ignored on a SignRequest] [Optional on a VerifyRequest]

357 As there may be multiple TransformedData / DocumentHash elements of the same document
358 having the same URI [RFC 2396] and RefType on a SignRequest or VerifyRequest - their
359 correspondance to an already existing <ds:Reference> however needs to be established on a
360 VerifyRequest only.

361 There is a need to disambiguate such cases. This Attribute hence offers a way to clearly identify the
362 <ds:Reference> when URI and RefType match multiple ds:References / TransformedData /
363 DocumentHash. The corresponding ds:Reference is indicated by this zero-based
364 WhichReference attribute (0 means the first <ds:Reference> in the signature, 1 means the
365 second, and so on).

366 Note: It may be possible to establish the ds:References / TransformedData / DocumentHash
367 correspondence by comparing the optionally supplied chain of transforms to those of the

368 ds:References having the same URI and RefType in the supplied ds:Signature if this chain of
369 transform has been supplied. This can be quite expensive and even out the advantages of
370 TransformedData / DocumentHash.

```
371 <xs:element name="TransformedData">  
372   <xs:complexType>  
373     <xs:complexContent>  
374       <xs:extension base="dss:DocumentBaseType">  
375         <xs:sequence>  
376           <xs:element ref="ds:Transforms" minOccurs="0"/>  
377           <xs:element ref="dss:Base64Data"/>  
378         </xs:sequence>  
379         <xs:attribute name="WhichReference" type="xs:integer"  
380                       use="optional"/>  
381       </xs:extension>  
382     </xs:complexContent>  
383   </xs:complexType>  
384 </xs:element>
```

385 2.4.4 Element <DocumentHash>

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

388 <ds:Transforms> [Required on a SignRequest] [Optional on VerifyRequest]

389 This specifies the value for a <ds:Reference> element's <ds:Transforms> child element when
390 referring to this document hash. In other words, this specifies transforms that the client has already
391 applied to the input document before hashing it.

392 <ds:DigestMethod> [Required on a SignRequest] [Optional on VerifyRequest]

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

396 <ds:DigestValue> [Required]

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

399 WhichReference [Ignored on a SignRequest] [Optional on a VerifyRequest]

400 As there may be multiple TransformedData / DocumentHash elements of the same document
401 having the same URI and RefType on a SignRequest or VerifyRequest - their correspondance
402 to an already existing <ds:Reference> however needs to be established on a VerifyRequest only.

403 There is a need to disambiguate such cases. This Attribute hence offers a way to clearly identify the
404 <ds:Reference> when URI and RefType match multiple ds:References / TransformedData /
405 DocumentHash. The corresponding ds:Reference is indicated by this zero-based
406 WhichReference attribute (0 means the first <ds:Reference> in the signature, 1 means the
407 second, and so on).

```
408 <xs:element name="DocumentHash">  
409   <xs:complexType>  
410     <xs:complexContent>  
411       <xs:extension base="dss:DocumentBaseType">  
412         <xs:sequence>  
413           <xs:element ref="ds:Transforms" minOccurs="0"/>  
414           <xs:element ref="ds:DigestMethod" minOccurs="0"/>  
415           <xs:element ref="ds:DigestValue"/>  
416         </xs:sequence>  
417         <xs:attribute name="WhichReference" type="xs:integer"  
418                       use="optional"/>  
419       </xs:extension>
```

420
421
422

```
</xs:complexContent>  
</xs:complexType>  
</xs:element>
```

423 2.5 Element <SignatureObject>

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

427 <ds:Signature> [Optional]

428 An XML signature [XMLDSIG].

429 <Timestamp> [Optional]

430 An XML, RFC 3161 or other timestamp (see section 5.1).

431 <Base64Signature> [Optional]

432 A base64 encoding of some non-XML signature, such as a PGP [RFC 2440] or CMS [RFC 3852]
433 signature. The type of signature is specified by its Type attribute (see section 7.1).

434 <SignaturePtr> [Optional]

435 This is used to point to an XML signature in an input (for a verify request) or output (for a sign
436 response) document in which a signature is enveloped.

437 SchemaRefs [Optional]

438 As described above in 2.4.1

439 A <SignaturePtr> contains the following attributes:

440 WhichDocument [Required]

441 This identifies the input document as in section 2.4.2 being pointed at (see also ID attribute in section
442 2.4.1).

443 XPath [Optional]

444 a) This identifies the signature element being pointed at.

445 b) The XPath expression is evaluated from the root node (see section 5.1 of [XPATH]) of the
446 document identified by WhichDocument after the XML data was extracted and parsed if necessary.
447 The context node for the XPath evaluation is the document's DocumentElement (see section 2.1 Well-
448 Formed XML Documents [XML]).

449 c) About namespace declarations for the expression necessary for evaluation see section 1 of
450 [XPATH]. Namespace prefixes used in XPath expressions MUST be declared within the element
451 containing the XPath expression. E.g.: <SignaturePtr
452 xmlns:ds="http://www.w3.org/2000/09/xmldsig#" XPath="//ds:Signature">. See
453 also the following example below. A piece of a XML signature of a <ds:Reference> containing a
454 <ds:Transforms> with a XPath filtering element that includes inline namespace prefixes
455 declaration. This piece of text comes from one of the signatures that were generated in the course of
456 the interoperability experimentation. As one can see they are added to the <ds:XPath> element:

```
457 <Reference URI="">  
458 <Transforms>  
459 <ds:Transform xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">  
460 <ds:XPath xmlns:upc1="http://www.ac.upc.edu/namespaces/ns1"  
461 xmlns:upc2="http://www.ac.upc.edu/namespaces/ns2">ancestor-or-  
462 self::upc1:Root</ds:XPath>  
463 </ds:Transform>  
464 </Transforms>  
465 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>  
466 <DigestValue>24xf8vfP3xJ40akfFAnEVM/zxXY=</DigestValue>
```


468 </Reference>

469 If the XPath does not evaluate to one element the server MUST return a <Result> (section 2.6)
470 issuing a <ResultMajor> RequesterError qualified by a <ResultMinor>
471 XPathEvaluationError.

472 <Other>

473 Other may contain arbitrary content that may be specified in a profile and can also be used to extend
474 the Protocol.

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

```
477 <xs:element name="SignatureObject">
478   <xs:complexType>
479     <xs:sequence>
480       <xs:choice>
481         <xs:element ref="ds:Signature"/>
482         <xs:element ref="dss:Timestamp"/>
483         <xs:element ref="dss:Base64Signature"/>
484         <xs:element ref="dss:SignaturePtr"/>
485         <xs:element name="Other" type="dss:AnyType"/>
486       </xs:choice>
487     </xs:sequence>
488     <xs:attribute name="SchemaRefs" type="xs:IDREFS" use="optional"/>
489   </xs:complexType>
490 </xs:element>
491 <xs:element name="Base64Signature">
492   <xs:complexType>
493     <xs:simpleContent>
494       <xs:extension base="xs:base64Binary">
495         <xs:attribute name="Type" type="xs:anyURI"/>
496       </xs:extension>
497     </xs:simpleContent>
498   </xs:complexType>
499 </xs:element>
500 <xs:element name="SignaturePtr">
501   <xs:complexType>
502     <xs:attribute name="WhichDocument" type="xs:IDREF"/>
503     <xs:attribute name="XPath" type="xs:string" use="optional"/>
504   </xs:complexType>
505 </xs:element>
```

506 2.6 Element <Result>

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

509 <ResultMajor> [Required]

510 The most significant component of the result code.

511 <ResultMinor> [Optional]

512 The least significant component of the result code.

513 <ResultMessage> [Optional]

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

```
515 <xs:element name="Result">
516   <xs:complexType>
517     <xs:sequence>
518       <xs:element name="ResultMajor" type="xs:anyURI"/>
519       <xs:element name="ResultMinor" type="xs:anyURI"
520         minOccurs="0"/>
521     </xs:sequence>
522 </xs:complexType>
523 </xs:element>
```

521
522
523
524
525

```
<xs:element name="ResultMessage"  
            type="dss:InternationalStringType" minOccurs="0"/>  
</xs:sequence>  
</xs:complexType>  
</xs:element>
```

526 The <ResultMajor> URIs MUST be values defined by this specification or by some profile of this
527 specification. The <ResultMajor> values defined by this specification are:

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

529 The protocol executed successfully.

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

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

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

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

534 urn:oasis:names:tc:dss:1.0:resultmajor:InsufficientInformation

535 The request could not be satisfied due to insufficient information.

536 In case of doubt of who is responsible a

537 urn:oasis:names:tc:dss:1.0:resultmajor:ResponderError is assumed.

538 This specification defines the following <ResultMinor> values, that are listed below, grouped by the
539 respective associated <ResultMajor> code.

540 One of the following <ResultMinor> values MUST be returned when the <ResultMajor> code is
541 Success.

542 urn:oasis:names:tc:dss:1.0:resultminor:valid:signature:OnAllDocuments

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

545 urn:oasis:names:tc:dss:1.0:resultminor:valid:signature:NotAllDocumentsReferen
546 ced

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

549 urn:oasis:names:tc:dss:1.0:resultminor:invalid:IncorrectSignature

550 The signature fails to verify, for example due to the signed document being modified or the incorrect
551 key being used.

552 urn:oasis:names:tc:dss:1.0:resultminor:valid:signature:HasManifestResults

553 The signature is valid with respect to XML Signature core validation. In addition, the message also
554 contains VerifyManifestResults.

555 Note: In the case that the core signature validation failed no attempt is made to verify the manifest.

556 urn:oasis:names:tc:dss:1.0:resultminor:valid:signature:InvalidSignatureTimest
557 amp

558 The signature is valid however the timestamp on that signature is invalid.

559 The following <ResultMinor> values is suggest MAY be returned when the <ResultMajor> code is
560 RequesterError.

561 urn:oasis:names:tc:dss:1.0:resultminor:ReferencedDocumentNotPresent

562 A ds:Reference element is present in the ds:Signature containing a full URI, but the corresponding
563 input document is not present in the request.

564 urn:oasis:names:tc:dss:1.0:resultminor:KeyInfoNotProvided

565 The required key information was not supplied by the client, but the server expected it to do so.

566 urn:oasis:names:tc:dss:1.0:resultminor:MoreThanOneRefUriOmitted

567 The server was not able to create a signature because more than one `RefUri` was omitted.
568 `urn:oasis:names:tc:dss:1.0:resultminor:InvalidRefURI`

569 The value of the `RefURI` attribute included in an input document is not valid.
570 `urn:oasis:names:tc:dss:1.0:resultminor:NotParseableXMLDocument`

571 The server was not able to parse a Document.
572 `urn:oasis:names:tc:dss:1.0:resultminor:NotSupported`

573 The server doesn't recognize or can't handle any optional input.
574 `urn:oasis:names:tc:dss:1.0:resultminor:Inappropriate:signature`

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

578 Further values for `<ResultMinor>` associated with `<ResultMajor>` code
579 `urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError` are left open to the implementer
580 or profile to be defined with in their namespaces.

581 The following `<ResultMinor>` values MAY be returned when the `<ResultMajor>` code is `ResponderError`.
582 `urn:oasis:names:tc:dss:1.0:resultminor:GeneralError`

583 The processing of the request failed due to an error not covered by the existing error codes. Further
584 details should be given in the result message for the user which may be passed on to the relevant
585 administrator.

586 `urn:oasis:names:tc:dss:1.0:resultminor:invalid:KeyLookupFailed`

587 Locating the identified key failed (e.g. look up failed in directory or in local key file).

588 Further values for `<ResultMinor>` associated with `<ResultMajor>` code
589 `urn:oasis:names:tc:dss:1.0:resultmajor:ResponderError` are left open to the implementer
590 or profile to be defined within their namespaces.

591 The following `<ResultMinor>` values MAY be returned when the `<ResultMajor>` code is
592 `InsufficientInformation`.

593 `urn:oasis:names:tc:dss:1.0:resultminor:CrlNotAvailiable`

594 The relevant certificate revocation list was not available for checking.
595 `urn:oasis:names:tc:dss:1.0:resultminor:OcspNotAvailiable`

596 The relevant revocation information was not available via the online certificate status protocol.
597 `urn:oasis:names:tc:dss:1.0:resultminor:CertificateChainNotComplete`

598 The chain of trust could not be established binding the public key used for validation to a trusted root
599 certification authority via potential intermediate certification authorities.

600 **2.7 Elements `<OptionalInputs>` and `<OptionalOutputs>`**

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

604 The `<OptionalInputs>` contains additional inputs associated with the processing of the request.
605 Profiles will specify the allowed optional inputs and their default values. The definition of an optional input
606 MAY include a default value, so that a client may omit the `<OptionalInputs>` yet still get service from
607 any profile-compliant DSS server.

608 If a server doesn't recognize or can't handle any optional input, it MUST reject the request with a
609 `<ResultMajor>` code of `RequesterError` and a `<ResultMinor>` code of `NotSupported` (see
610 section 2.6).

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

614 The <OptionalInputs> and <OptionalOutputs> elements contain unordered inputs and outputs.
615 Applications MUST be able to handle optional inputs or outputs appearing in any order within these
616 elements. Normally, there will only be at most one occurrence of any particular optional input or output
617 within a protocol message. Where multiple occurrences of an optional input (e.g. <IncludeObject> in
618 section 3.5.6) or optional output are allowed, it will be explicitly specified (see section 4.5.9 for an
619 example).

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

```
621 <xs:element name="OptionalInputs" type="dss:AnyType" />  
622  
623 <xs:element name="OptionalOutputs" type="dss:AnyType" />
```

624 2.8 Common Optional Inputs

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

626 2.8.1 Optional Input <ServicePolicy>

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

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

633 2.8.2 Optional Input <ClaimedIdentity>

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

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

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

```
645 <xs:element name="ClaimedIdentity">  
646   <xs:complexType>  
647     <xs:sequence>  
648       <xs:element name="Name" type="saml:NameIdentifierType" />  
649       <xs:element name="SupportingInfo" type="dss:AnyType"  
650         minOccurs="0" />  
651     </xs:sequence>  
652   </xs:complexType>  
653 </xs:element>
```

654 2.8.3 Optional Input <Language>

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

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

658 2.8.4 Optional Input <AdditionalProfile>

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

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

664 2.8.5 Optional Input <Schemas>

665 The <Schemas> element provides an in band mechanism for communicating XML schemas required for
666 validating an XML document.

```
667 <xs:element name="Schemas" type="dss:SchemasType" />  
668 <xs:complexType name="SchemasType">  
669   <xs:sequence>  
670     <xs:element ref="dss:Schema" minOccurs="1" maxOccurs="unbounded" />  
671   </xs:sequence>  
672 </xs:complexType>  
673  
674 <xs:element name="Schema" type="dss:DocumentType" />
```

675 An XML schema is itself an XML document, however, only the following attributes, defined in
676 dss:DocumentType, are meaningful for the <Schema> element:

677 ID

678 Used by relying XML document to identify a schema.

679 RefURI

680 The target namespace of the schema (i.e. the value of the targetNamespace attribute).

681 RefType

682 MUST NOT be used.

683 SchemaRefs

684 MUST NOT be used.

685 Note: It is recommended to use xml:id as defined in [xml:id] as id in the payload being referenced by a
686 <ds:Reference>, because the schema then does not have to be supplied for identifying the ID
687 attributes.

688 2.9 Common Optional Outputs

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

690 2.9.1 Optional Output <Schemas>

691 The <Schemas> element is typically used as an optional input in a <VerifyRequest>. However, there
692 are situations where it may be used as an optional output. For example, a service that makes use of the
693 <ReturnUpdatedSignature> mechanism may, after verifying a signature over an input document,
694 generate a signature over a document of a different schema than the input document. In this case the
695 <Schemas> element MAY be used to communicate the XML schemas required for validating the returned
696 XML document.

697 For a description of the <Schemas> element see section 2.8.5.

698 2.10 Type <RequestBaseType>

699 The <RequestBaseType> complex type is the base structure for request elements defined by the core
700 protocol or profiles. It defines the following attributes and elements:

701 RequestID [Optional]

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

704 Profile [Optional]

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

708 <OptionalInputs> [Optional]

709 Any additional inputs to the request.

710 <InputDocuments> [Optional]

711 The input documents which the processing will be applied to.

```
712 <xs:complexType name="RequestBaseType">  
713   <xs:sequence>  
714     <xs:element ref="dss:OptionalInputs" minOccurs="0"/>  
715     <xs:element ref="dss:InputDocuments" minOccurs="0"/>  
716   </xs:sequence>  
717   <xs:attribute name="RequestID" type="xs:string"  
718     use="optional"/>  
719   <xs:attribute name="Profile" type="xs:anyURI" use="optional"/>  
720 </xs:complexType>
```

721 2.11 Type <ResponseBaseType>

722 The <ResponseBaseType> complex type is the base structure for response elements defined by the
723 core protocol or profiles. It defines the following attributes and elements:

724 RequestID [Optional]

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

727 Profile [Required]

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

730 <Result> [Required]

731 A code representing the status of the request.

732 <OptionalOutputs> [Optional]

733 Any additional outputs returned by the server.

```
734 <xs:complexType name="ResponseBaseType">  
735   <xs:sequence>  
736     <xs:element ref="dss:Result"/>  
737     <xs:element ref="dss:OptionalOutputs" minOccurs="0"/>  
738   </xs:sequence>  
739   <xs:attribute name="RequestID" type="xs:string"  
740     use="optional"/>  
741   <xs:attribute name="Profile" type="xs:anyURI" use="required"/>  
742 </xs:complexType>
```

743 2.12 Element <Response>

744 The <Response> element is an instance of the <ResponseBaseType> type. This element is useful in
745 cases where the DSS server is not able to respond with a special response type. It is a general purpose
746 response element for exceptional circumstances.

747 E.g.: "The server only supports verification requests.", "The server is currently under maintenance" or
748 "The service operates from 8:00 to 17:00".

749 Other use cases for this type are expected to be described in special profiles (e.g. the Asynchronous
750 profile).

751 `<xs:element name="Response" type="dss:ResponseBaseType"/>`

752

3 The DSS Signing Protocol

3.1 Element <SignRequest>

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

756 RequestID [Optional]

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

759 Profile [Optional]

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

763 <OptionalInputs> [Optional]

764 Any additional inputs to the request.

765 <InputDocuments> [Optional]

766 The input documents, which the signature will be calculated over. This element, while optional in
767 RequestBaseType, is REQUIRED for the <SignRequest> element.

```
768 <xs:element name="SignRequest" >  
769   <xs:complexType>  
770     <xs:complexContent>  
771       <xs:extension base="dss:RequestBaseType" />  
772     </xs:complexContent>  
773   </xs:complexType>  
774 </xs:element>
```

3.2 Element <SignResponse>

776 The <SignResponse> element contains the following attributes and elements inherited from
777 <ResponseBaseType>:

778 RequestID [Optional]

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

781 Profile [Optional]

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

784 <Result> [Required]

785 A code representing the status of the request.

786 <OptionalOutputs> [Optional]

787 Any additional outputs returned by the server.

788 In addition to <ResponseBaseType> the <SignResponse> element defines the following

789 <SignatureObject> element:

790 <SignatureObject> [Optional]

791 The result signature or timestamp or, in the case of a signature being enveloped in an output
792 document (see section 3.5.8), a pointer to the signature.

793 In the case of <SignaturePlacement> being used this MUST contain a <SignaturePtr>, having
794 the same XPath expression as in <SignaturePlacement> and pointing to a
795 <DocumentWithSignature> using it's WhichDocument attribute.

```
796 <xs:element name="SignResponse">  
797   <xs:complexType>  
798     <xs:complexContent>  
799       <xs:extension base="dss:ResponseBaseType">  
800         <xs:sequence>  
801           <xs:element ref="dss:SignatureObject" minOccurs="0"/>  
802         </xs:sequence>  
803       </xs:extension>  
804     </xs:complexContent>  
805   </xs:complexType>  
806 </xs:element>
```

807 3.3 Processing for XML Signatures

808 3.3.1 Basic Process for <Base64XML>

809 A DSS server that produces XML signatures SHOULD perform the following steps, upon receiving a
810 <SignRequest>.

811 These steps may be changed or overridden by procedures defined for the optional inputs (for example,
812 see section 3.5.6), or by the profile or policy the server is operating under.

813 The ordering of the <Document> elements inside the <InputDocuments> MAY be ignored by the
814 server.

815 1. For each <Document> in <InputDocuments> the server MUST perform the following steps:

816 a. In the case of <Base64XML> (see later sub-sections for other cases), the server base64-
817 decodes the data contained within <Document> into an octet stream. This data MUST
818 be a well formed XML Document as defined in [XML] section 2.1. If the RefURI attribute
819 references within the same input document then the server parses the octet stream to
820 NodeSetData (see [XMLDSIG] section 4.3.3.3) before proceeding to the next step.

821 b. The data is processed and transforms applied by the server to produce a canonicalized
822 octet string as required in [XMLDSIG] section 4.3.3.2.

823 Note: Transforms can be applied as a server implementation MAY choose to increase
824 robustness of the Signatures created. These Transforms may reflect idiosyncrasies of
825 different parsers or solve encoding issues or the like. Servers MAY choose not to apply
826 transforms in basic processing and extract the binary data for direct hashing or
827 canonicalize the data directly if certain optional inputs (see sections 3.5.8 point 2 and d.v,
828 3.5.9) are not to be implemented.

829 Note: As required in [XMLDSIG] if the end result is an XML node set, the server MUST
830 attempt to convert the node set back into an octet stream using Canonical XML [XML-
831 C14N].

832 c. The hash of the resulting octet stream is calculated.

833 d. The server forms a <ds:Reference> with the elements and attributes set as follows:

834 i. If the <Document> has a RefURI attribute, the <ds:Reference> element's
835 URI attribute is set to the value of the RefURI attribute, else this attribute is
836 omitted.

837 A signature MUST NOT be created if more than one RefURI is omitted in the set
838 of input documents and the server MUST report a RequesterError by setting
839 <ResultMajor> RequesterError qualified by a <ResultMinor>.

840 ii. If the <Document> has a RefType attribute, the <ds:Reference> element's
841 Type attribute is set to the value of the RefType attribute, else this attribute is
842 omitted.

- 843 iii. The `<ds:DigestMethod>` element is set to the hash method used.
- 844 iv. The `<ds:DigestValue>` element is set to the hash value that is to be
845 calculated as per **[XMLDSIG]**.
- 846 v. The `<ds:Transforms>` element is set to the sequence of transforms applied by
847 the server in step b. This sequence **MUST** describe the effective transform as a
848 reproducible procedure from parsing until hash.
- 849 2. References resulting from processing of optional inputs **MUST** be included. In doing so, the server
850 **MAY** reflect the ordering of the `<Document>` elements.
- 851 3. The server creates an XML signature using the `<ds:Reference>` elements created in Step 1.d,
852 according to the processing rules in **[XMLDSIG]**.

853 **3.3.2 Process Variant for `<InlineXML>`**

854 In the case of an input document which contains `<InlineXML>` Step 3.3.1 1.a is replaced with the
855 following step:

- 856 1.
- 857 a. The XML document is extracted from the DSS protocol envelope, without taking inherited
858 namespaces and attributes. Exclusive Canonical XML **[XML-xcl-c14n]** **MUST** be applied to
859 extract data **AND** assure context free extraction.
860 If signed data is to be echoed back to the client and hence details could get lost refer to
861 **Error! Reference source not found.**

862 In Step 3.3.1 step 1.d.v, the `<ds:Transforms>` element **MUST** begin with the canonicalization transform
863 applied under revised step 3.3.2 1.a above.

864 **3.3.3 Process Variant for `<EscapedXML>`**

865 In the case of an input document which contains `<EscapedXML>` Step 3.3.1 1.a is replaced with the
866 following:

- 867 1.
- 868 In the case of `<EscapedXML>` the server unescapes the data contained within `<Document>` into
869 a character string. If the RefURI references within the same input document the server
870 parses the unescaped character content to NodeSetData if necessary. If the RefURI does not
871 reference within the same input document then the server canonicalizes the characters or
872 parsed NodeSetData (see **[XMLDSIG]** section 4.3.3.3) to octet stream if necessary before
873 proceeding to the next step.
- 874 Note: If the characters are converted to an octet stream directly a consistent encoding
875 including ByteOrderMark has to be ensured.

877 In Step 3.3.1 1.d.v, the `<ds:Transforms>` element **MUST** begin with the canonicalization transform
878 applied under revised step 3.3.3 0 above.

879 **3.3.4 Process Variant for `<Base64Data>`**

880 In the case of an input document which contains `<Base64data>` Step 1 a and Step 1 b are replaced
881 with the following:

- 882 1.
- 883 a. The server base64-decodes the data contained within `<Document>` into an octet string.
- 884 b. No transforms or other changes are made to the octet string before hashing.
- 885 Note: If the RefURI references within the same input document the Document **MUST** also be
886 referenced by `<IncludeObject>` in section 3.5.6 to include the object as base64 data
887

888 inside a <ds:Object> otherwise a <Result> (section 2.6) issuing a <ResultMajor>
889 RequesterError qualified by a <ResultMinor> NotParseableXMLDocument.

890 **3.3.5 Process Variant for <TransformedData>**

891 In the case of an input document which contains <TransformedData> Step 3.3.1 1 is replaced with the
892 following:

893 1. For each <TransformedData> in <InputDocuments> the server MUST perform the following
894 steps:

895 a. The server base64-decodes the data contained within <Base64Data> of
896 <TransformedData> into an octet string.

897 b. Omitted.

898 c. The hash over of the octet stream extracted in step a is calculated.

899 d. as in 3.3.1 step 1d updated as follows

900 replace the word "<Document>" by <TransformedData> otherwise as in as 3.3.1
901 step 1d.i.

902 replace the word "<Document>" by <TransformedData> otherwise as in as 3.3.1
903 step 1d.ii.

904 same as 3.3.1 step 1d.iii.

905 The <ds:Transforms> element is set to the sequence of transforms indicated by the
906 client in the <ds:Transforms> element within the <TransformedData>. This
907 sequence MUST describe the effective transform as a reproducible procedure from
908 parsing until digest input.

909 **3.3.6 Process Variant for <DocumentHash>**

910 In the case of an input document which is provided in the form of a hash value in <DocumentHash> Step
911 3.3.1 1 is replaced with the following:

912 1. For each <DocumentHash> in <InputDocuments> the server MUST perform the following steps:

913 a. Omitted.

914 b. Omitted.

915 c. Omitted.

916 d. as in 3.3.1 step 1d updated as follows

917 i. replace the word "<Document>" by <DocumentHash> otherwise as in as 3.3.1 step
918 1d.i.

919 ii. replace the word "<Document>" by <DocumentHash> otherwise as in as 3.3.1 step
920 1d.ii.

921 iii. The <ds:DigestMethod> element is set to the value of <ds:DigestMethod> in
922 <DocumentHash>

923 iv. The <ds:DigestValue> element is set to the value of <ds:DigestValue> in
924 <DocumentHash>.

925 v. The <ds:Transforms> element is set to the sequence of transforms indicated by
926 the client in the <ds:Transforms> element within <DocumentHash>, if any such
927 transforms are indicated by the client. This sequence MUST describe the effective
928 transform as a reproducible procedure from parsing until hash.

929 **3.4 Basic Processing for CMS Signatures**

930 A DSS server that produces CMS signatures [RFC 3852] SHOULD perform the following steps, upon
931 receiving a <SignRequest>. These steps may be changed or overridden by the optional inputs, or by
932 the profile or policy the server is operating under. With regard to the compatibility issues in validation /
933 integration of PKCS#7 signatures and CMS implementations please refer to [RFC 3852] section 1.1.1
934 "Changes Since PKCS #7 Version 1.5".

935 The <SignRequest> MUST contain either a single <Document> not having RefURI, RefType set or
936 a single <DocumentHash> not having RefURI, RefType, <ds:Transforms> set:

937 1. If a <Document> is present, the server hashes its contents as follows:

938 a. If the <Document> contains <Base64XML>, the server extracts the ancestry context free text
939 content of the <Base64XML> as an octet stream by base64 decoding it's contents.

940 b. If the <Document> contains <InlineXML>, the server extracts the ancestry context free text
941 content of the <InlineXML> as an octet stream as explained in (section 3.3.2 1.a). This
942 octet stream has to be returned as <TransformedDocument>/ <Base64XML>. For CMS
943 signatures this only has to be returned in the case of CMS signatures that are
944 external/detached/"without eContent", as these return the signed Data anyway.

945 c. If the <Document> contains <EscapedXML>, the server unescapes the content of the
946 <EscapedXML> as a character stream and converts the character stream to an octet stream
947 using an encoding as explained in (section 3.3.3).

948 d. If the <Document> contains <Base64Data>, the server base64-decodes the text content of
949 the <Base64Data> into an octet stream.

950 e. The server hashes the resultant octet stream.

951 2. The server forms a SignerInfo structure based on the input document. The components of the
952 SignerInfo are set as follows:

953 a. The digestAlgorithm field is set to the OID value for the hash method that was used in
954 step 1.c (for a <Document>), or to the OID value that is equivalent to the input document's
955 <ds:DigestMethod> (for a <DocumentHash>).

956 b. The signedAttributes field's message-digest attribute contains the hash value that was
957 calculated in step 1.e (for a <Document>), or that was sent in the input document's
958 <ds:DigestValue> (for a <DocumentHash>). Other signedAttributes may be added
959 by the server, according to its profile or policy, or according to the <Properties> optional
960 input (see section 3.5.5).

961 c. The remaining fields (sid, signatureAlgorithm, and signature) are filled in as per a
962 normal CMS signature.

963 3. The server creates a CMS signature (i.e. a SignedData structure) containing the SignerInfo that
964 was created in Step 2. The resulting SignedData should be detached (i.e. external or "without
965 eContent") unless the client sends the <IncludeEContent> optional input (see section 3.5.9).

966 **3.4.1 Process Variant for <DocumentHash>**

967 In the case of a <DocumentHash> the processing by the server is as follows:

968 1. Omitted.

969 a. Omitted.

970 b. Omitted.

971 c. Omitted.

972 d. Omitted.

973 e. Omitted.

974 2. Same as in 3.4 step 2

- 975 a. Unchanged.
976 b. Unchanged.
977 c. Unchanged.
978 3. As in 3.4 step 3, with the requirement that the signature has to be external/detached/"without
979 eContent", since <DocumentHash> is incompatible with optional input <IncludeEContent> (see
980 3.5.7).

981 3.5 Optional Inputs and Outputs

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

986 3.5.1 Optional Input <SignatureType>

987 The <SignatureType> element indicates the type of signature or timestamp to produce (such as a XML
988 signature, a XML timestamp, a RFC 3161 timestamp, a CMS signature, etc.). See section 7.1 for some
989 URI references that MAY be used as the value of this element.

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

991 3.5.2 Optional Input <AddTimestamp>

992 The <AddTimestamp> element indicates that the client wishes the server to embed a timestamp token
993 as a property or attribute of the resultant or the supplied signature. The timestamp token will be applied to
994 the signature value in the case of CMS/PKCS7 signatures or the <ds:SignatureValue> element in the
995 case of XML signatures.

996 Note: Procedures for handling other forms of timestamp may be defined in profiles of the Core. In
997 particular, the DSS AdES profile **[DSS-AdES-P]** defines procedures for generating timestamps over the
998 content which is about to be signed (sometimes called content timestamps), and the DSS Timestamp
999 profile **[DSS-TS-P]** defines procedures for handling standalone timestamps.

1000 The schema definition of this optional input is as follows:

```
1001 <xs:element name="AddTimestamp" type="dss:UpdateSignatureInstructionType" />  
1002 <xs:complexType name="TimeSignatureInstructionType">  
1003 <xs:complexContent>  
1004 <xs:extension base="dss:UpdateSignatureInstructionType">  
1005 <xs:attribute name="TimeStampTheGivenSignature" type="xs:boolean"  
1006 use="optional" default="false" />  
1007 </xs:extension>  
1008 </xs:complexContent>  
1009 </xs:complexType>
```

1010 The type UpdateSignatureInstructionType is defined as follows:

```
1011 <xs:complexType name="UpdateSignatureInstructionType">  
1012 <xs:attribute name="Type" type="xs:anyURI" use="optional" />  
1013 </xs:complexType>
```

1014 The Type attribute, if present, indicates what type of timestamp to apply. Profiles that use this optional
1015 input MUST define the allowed values, and the default value, for the Type attribute (unless only a single
1016 type of timestamp is supported, in which case the Type attribute can be omitted).

1017 Two scenarios for the timestamping of both CMS and XML signatures are supported by this Optional
1018 Input. They are as follows:

- 1019 a) Create and embed a timestamp token into the signature being created as part of this SignRequest .

1020 b) Create and embed a timestamp token into an existing signature, without verification, which is passed in
1021 the <InputDocuments> element of this SignRequest.

1022 The following subsections specify the use of RFC 3161 timestamps with CMS signatures and the use of
1023 XML Timestamps or RFC 3161 timestamps with XML Signature. These subsections address both
1024 scenarios.

1025 **3.5.2.1 Processing for CMS signatures time-stamping**

1026 In both scenarios, the timestamp token created by the server SHALL be created according to
1027 **[RFC 3161]**. The MessageImprint field within the TstInfo structure of the timestamp token will be
1028 derived from the signature value of the just-created or incoming signature depending on the scenario.
1029 The timestamp SHALL be embedded in the CMS signature as an unsigned attribute with the object
1030 identifier (see Appendix A of **[RFC 3161]**):

1031 { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 14}

1032 The signature and its embedded timestamp is returned in the <SignatureObject> of the
1033 <SignResponse>.

1034 In scenario b) the incoming signature is passed in a <Base64Data> element, with the MimeType
1035 attribute set to application/pkcs7-signature.

1036 The Type attribute of the <AddTimestamp> optional input SHALL be set to:

1037 "urn:ietf:rfc:3161".

1038 Note: In scenario b) the server SHOULD not verify the signature before adding the timestamp. If a client
1039 wishes that its signatures be verified as a condition of time stamping, the client SHOULD use the
1040 <AddTimestamp> optional input of the Verify protocol.

1041 **3.5.2.2 Processing for XML Timestamps on XML signatures**

1042 If the type attribute in this optional input is
1043 urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken and signature being
1044 timestamped is an XML signature, then the XML signature MUST contain <dss:timestamp> as defined
1045 in 5.1, placed in a <xades:XMLTimestamp> within a
1046 <xades:SignatureTimeStamp> as defined in **[XAdES]**.

1047 The <dss:timestamp> MUST contain <ds:Signature> with at least two <ds:Reference>
1048 elements:

1049 - One with the Type attribute set to
1050 "urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken". and referencing a
1051 <ds:Object> element whose content is a <TSTInfo> element.

1052 - The other referencing the <ds:SignatureValue> being timestamped.

1053 The present specification defines a format for XML timestamp tokens. In addition XAdES defines a
1054 mechanism for incorporating signature timestamps in XML signatures. The present document mandates
1055 that signature timestamps in XML format MUST follow the syntax defined in section 5.1 of this document.
1056 These time-stamp tokens MUST be added to XML signatures as specified by XAdES.

1057 The signature and its embedded timestamp SHALL be returned in the <SignatureObject> of the
1058 <SignResponse>.

1059 In scenario b) the incoming signature MUST be passed in on one of the following three elements
1060 <EscapedXML>, <InlineXML> or <Base64XML>.

1061 Note: In scenario b) the server SHOULD not verify the signature before adding the timestamp. If a client
1062 wishes that its signatures be verified as a condition of time stamping, the client SHOULD use the
1063 <AddTimestamp> optional input of the Verify protocol.

1064 The Type attribute of the <AddTimestamp> optional input SHALL be set to:

1065 "urn: oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken".

1066 3.5.2.3 Processing for RFC 3161 Timestamps on XML signatures

1067 If the type attribute in this optional input is `urn:ietf:rfc:3161` and signature being timestamped is an
1068 XML signature then the XML signature MUST contain an RFC 3161, placed in a
1069 `<xades:EncapsulatedTimeStamp>` within a `<xades:SignatureTimeStamp>` as defined in
1070 [XAdES].

1071 In scenario b) the incoming signature MUST be passed in on one of the following three elements
1072 `<EscapedXML>`, `<InlineXML>` or `<Base64XML>`.

1073 Note: In scenario b) the server SHOULD not verify the signature before adding the timestamp. If a client
1074 wishes that its signatures be verified as a condition of time stamping, the client SHOULD use the
1075 `<AddTimeStamp>` optional input of the Verify protocol.

1076 3.5.3 Optional Input `<IntendedAudience>`

1077 The `<IntendedAudience>` element tells the server who the target audience of this signature is. The
1078 server MAY use this to parameterize any aspect of its processing (for example, the server MAY choose to
1079 sign with a key that it knows a particular recipient trusts).

```
1080 <xs:element name="IntendedAudience">  
1081   <xs:complexType>  
1082     <xs:sequence>  
1083       <xs:element name="Recipient" type="saml:NameIdentifierType"  
1084         maxOccurs="unbounded" />  
1085     </xs:sequence>  
1086   </xs:complexType>  
1087 </xs:element>
```

1088 3.5.4 Optional Input `<KeySelector>`

1089 The `<KeySelector>` element tells the server which key to use.

```
1090 <xs:element name="KeySelector">  
1091   <xs:complexType>  
1092     <xs:choice>  
1093       <xs:element ref="ds:KeyInfo" />  
1094       <xs:element name="Other" type="dss:AnyType" />  
1095     </xs:choice>  
1096   </xs:complexType>  
1097 </xs:element>
```

1098 3.5.5 Optional Input `<Properties>`

1099 The `<Properties>` element is used to request that the server add certain signed or unsigned properties
1100 (aka "signature attributes") into the signature. The client can send the server a particular value to use for
1101 each property, or leave the value up to the server to determine. The server can add additional properties,
1102 even if these aren't requested by the client.

1103 The `<Properties>` element contains:

1104 `<SignedProperties>` [Optional]

1105 These properties will be covered by the signature.

1106 `<UnsignedProperties>` [Optional]

1107 These properties will not be covered by the signature.

1108 Each `<Property>` element contains:

1109 `<Identifier>` [Required]

1110 A URI reference identifying the property.

1111 `<Value>` [Optional]

1112 If present, the value the server should use for the property.
1113 This specification does not define any properties. Profiles that make use of this element MUST define the
1114 allowed property URIs and their allowed values.

```
1115 <xs:element name="Properties">  
1116 <xs:complexType>  
1117 <xs:sequence>  
1118 <xs:element name="SignedProperties"  
1119 type="dss:PropertiesType" minOccurs="0"/>  
1120 <xs:element name="UnsignedProperties"  
1121 type="dss: PropertiesType" minOccurs="0"/>  
1122 </xs:sequence>  
1123 </xs:complexType>  
1124 </xs:element>  
1125  
1126 <xs:complexType name="PropertiesType">  
1127 <xs:sequence>  
1128 <xs:element ref="dss:Property" maxOccurs="unbounded"/>  
1129 </xs:sequence>  
1130 </xs:complexType>  
1131  
1132 <xs:element name="Property">  
1133 <xs:complexType>  
1134 <xs:sequence>  
1135 <xs:element name="Identifier" type="xs:anyURI"/>  
1136 <xs:element name="Value" type="dss:AnyType"  
1137 minOccurs="0"/>  
1138 </xs:sequence>  
1139 </xs:complexType>  
1140 </xs:element>
```

1141 3.5.6 Optional Input <IncludeObject>

1142 Optional input <IncludeObject> is used to request the creation of an XMLSig enveloping signature as
1143 follows. Multiple occurrences of this optional input can be present in a single <SignRequest> message.
1144 Each occurrence will cause the inclusion of an object inside the signature being created.

1145 The attributes of <IncludeObject> are:

1146 whichDocument [Required]

1147 Identifies the input document which will be inserted into the returned signature (see the ID attribute in
1148 section 2.4.1).

1149 hasObjectTagsAndAttributesSet

1150 If True indicates that the <Document> contains a <ds:Object> element which has been prepared
1151 ready for direct inclusion in the <ds:Signature>.

1152 objId [optional]

1153 Sets the Id attribute on the returned <ds:Object>.

1154 createReference

1155 This attribute set to false inhibits the creation, carried by the Basic Processing specified in section
1156 3.3.1, of the <ds:Reference> associated to the RefURI attribute of the input document referred by
1157 the whichDocument attribute, effectively allowing clients to include <ds:Object> elements not
1158 covered/protected by the signature being created.

```
1159 <xs:element name="IncludeObject">  
1160 <xs:complexType>  
1161 <xs:attribute name="whichDocument" type="xs:IDREF"/>  
1162 <xs:attribute name="hasObjectTagsAndAttributesSet"  
1163 type="xs:boolean" default="false"/>
```



```

1164     <xs:attribute name="ObjId" type="xs:string"
1165               use="optional" />
1166     <xs:attribute name="createReference" type="xs:boolean"
1167               use="optional" default="true" />
1168   </xs:complexType>
1169 </xs:element>

```

1170 3.5.6.1 XML Signatures Variant Optional Input <IncludeObject>

1171 An enveloping signature is a signature having <ds:Object>s which are referenced by
 1172 <ds:Reference>s having a same-document URI.

1173 For each <IncludeObject> the server creates a new <ds:Object> element containing the document,
 1174 as identified using the WhichDocument attribute, as its child. This object is carried within the enveloping
 1175 signature. The ordering of the <IncludeObject> optional inputs MAY be ignored by the server.

1176 This <Document> MUST include a "same-document" RefURI attribute (having a value starting with "#")
 1177 which references either:

- 1178 • *The whole newly-created <ds:Object>.*
- 1179 • *The relevant parts of the newly-created <ds:Object>'s contents to be covered/protected by the*
 1180 *signature (only applicable when the <Document> element contains either <Base64XML>, <InlineXML> or <EscapedXML>)*

1182 If the result of evaluating the expression included in the RefURI attribute doesn't fit in any of the options
 1183 described above, the server MUST reject the request using a <ResultMajor> RequesterError which
 1184 MAY be qualified by a <ResultMinor>
 1185 urn:oasis:names:tc:dss:1.0:resultminor:InvalidRefURI

1186 Note :If the server does not support the ordering of <ds:Object>, it is recommended either to use ID-
 1187 based referencing to the <ds:Object> (using the client-generated ID included in the ObjId attribute) or
 1188 to rely on expressions based on <ds:Object>'s contents that allow to unambiguously refer to the
 1189 included object or their relevant parts.

1190 The URI in the RefURI attribute of this <Document> should at least reference the relevant parts of the
 1191 Object to be included in the calculation for the corresponding reference. Clients MUST generate requests
 1192 in a way that some <ds:Reference>'s URI values actually will reference the <ds:Object> generated
 1193 by the server once this element will have been included in the <ds:Signature> produced by the server.

- 1194 1. For each <IncludeObject> the server MUST carry out the following steps before performing Basic
 1195 Processing (as specified in section 3.3.1):
 - 1196 a. The server identifies the <Document> that is to be placed into a <ds:Object> as indicated by
 1197 the WhichDocument attribute.
 - 1198 b. The data to be carried in the enveloping signature is extracted and decoded as described in
 1199 3.3.1 Step 1 a (or equivalent step in variants of the basic process as defined in 3.3.2 onwards
 1200 depending of the form of the input document).
 - 1201 c. if the hasObjectTagsAndAttributesSet attribute is false or not present the server builds the
 1202 <ds:Object> as follows:
 - 1203 i. The server generates the new <ds:Object> and sets its Id attribute to the value
 1204 indicated in ObjId attribute of the optional input if present.
 - 1205 ii. In the case of the Document pointed at by WhichDocument having Base64Data,
 1206 <ds:Object>'s MIME Type is to be set to the value of <dss:Base64Data>'s MIME
 1207 Type value and the Encoding is to be set to <http://www.w3.org/TR/xmlschema-2/#base64Binary>
 1208
 - 1209 d. The server splices the to-be-enveloped documents as <ds:Object>(s) into the
 1210 <ds:Signature>, which is to be returned.
 - 1211 e. If CreateReference is set to true generate a ds:Reference element referencing the
 1212 spliced <ds:Object> and exclude this <Document> from the set of <Document>s ready for

1213 further processing. Otherwise just exclude this <Document> from the set of <Document>s
1214 ready for further processing.
1215 2. The server then continues with processing as specified in section 3.3.1 for the rest of the documents.

1216 3.5.7 Optional Input <IncludeEContent>

1217 In the case of the optional input <IncludeEContent> (that stands for include enveloped or
1218 encapsulated content) section 3.4 step 3 is overridden as follows.

1219 3. The server creates a CMS signature (i.e. a SignedData structure) containing the SignerInfo that
1220 was created in Step 3. The resulting SignedData is now internal, as the document is enveloped in
1221 the signature.

1222 For CMS details in this context please refer to [RFC 3852] sections 5.1 "SignedData Type" and 5.2
1223 "EncapsulatedContentInfo Type".

1224 3.5.8 Enveloped Signatures, Optional Input <SignaturePlacement> and 1225 Output <DocumentWithSignature>

1226 Optional input <SignaturePlacement> is used to request the creation of an XMLSig enveloped
1227 signature placed within an input document. The resulting document with the enveloped signature is
1228 placed in the optional output <DocumentWithSignature>.

1229 The server places the signature in the document identified using the WhichDocument attribute.

1230 In the case of a non-XML input document then the server will return an error unless alternative
1231 procedures are defined by a profile or in the server policy for handling such a situation.

1232 The <SignaturePlacement> element contains the following attributes and elements:

1233 WhichDocument [Required]

1234 Identifies the input document which the signature will be inserted into (see the ID attribute in section
1235 2.4.1).

1236 CreateEnvelopedSignature

1237 If this is set to true a reference having an enveloped signature transform is created.

1238 <XPathAfter> [Optional]

1239 Identifies an element, inside the XML input document, after which the signature will be inserted. (The
1240 rules for XPath evaluation are those stated in section 2.5 SignatureObject)

1241 <XPathFirstChildOf> [Optional]

1242 Identifies an element, in the XML input document, which the signature will be inserted as the first child
1243 of. For details on the evaluation of The XPath expression see above (<XPathAfter>). The
1244 signature is placed immediately after the start tag of the specified element.

```
1245 <xs:element name="SignaturePlacement">  
1246 <xs:complexType>  
1247 <xs:choice>  
1248 <xs:element name="XPathAfter" type="xs:string"/>  
1249 <xs:element name="XPathFirstChildOf"  
1250 type="xs:string"/>  
1251 </xs:choice>  
1252 <xs:attribute name="WhichDocument" type="xs:IDREF"/>  
1253 <xs:attribute name="CreateEnvelopedSignature"  
1254 type="xs:boolean" default="true"/>  
1255 </xs:complexType>  
1256 </xs:element>
```

1257 The <DocumentWithSignature> optional output contains the input document with the signature
1258 inserted. It has one child element:

1259 <Document> [Required]

1260 This contains the input document with a signature inserted in some fashion.

```
1261 <xs:element name="DocumentWithSignature">
1262   <xs:complexType>
1263     <xs:sequence>
1264       <xs:element ref="dss:Document" />
1265     <xs:sequence>
1266   </xs:complexType>
1267 </xs:element>
```

1268 For an XMLSig enveloped signature the client produces a request including elements set as follows:

- 1269 1. The WhichDocument attribute is set to identify the <Document> to envelope the signature.
- 1270 2. The RefURI attribute MUST be set to include a "same-document" URI which references either:
- 1271 - The whole <Document> containing the signature (by using a RefURI="")
- 1272 - The relevant parts of the <Document> to be covered/protected by the signature (by using a "same-
- 1273 document" RefURI attribute having a value starting with "#", like RefURI="#some-id",
- 1274 RefURI="#xpointer()", RefURI="#xpointer(/DocumentElement/ToBeSignedElement)" or the like).
- 1275 If the result of evaluating the expression included in the RefURI attribute doesn't fit in any of the
- 1276 options described above, the server MUST reject the request using a <ResultMajor>
- 1277 RequesterError which MAY be qualified by a <ResultMinor>
- 1278 urn:oasis:names:tc:dss:1.0:resultminor:InvalidRefURI.
- 1279 3. The createEnvelopedSignature is set to true (or simply omitted).

1280 If the <SignaturePlacement> element is present the server processes it as follows before performing

1281 Basic Processing (as specified in section 3.3.1):

- 1282 1. The server identifies the <Document> in which the signature is to be enveloped as indicated by the
- 1283 WhichDocument attribute.
- 1284 2. This document is extracted and decoded as described in 3.3.1 Step 1.a (or equivalent step in variants
- 1285 of the basic process as defined in 3.3.2 onwards depending of the form of the input document).
- 1286 3. The server splices the <ds:Signature> to-be-enveloped into the document.
- 1287 4. If createEnvelopedSignature equals true,
- 1288 a. Perform Basic Processing for the enveloping <Document>, as described in section 3.3.1 with the
- 1289 following amendments:
- 1290 1.
- 1291 a. Omitted
- 1292 b. As in 3.3.1 1.b, with the additional requirement of adding an
- 1293 EnvelopedSignatureTransform as the first transform in the <ds:Transforms> list
- 1294 (even preceding transforms used for extraction).
- 1295 Note: This is necessary because the EnvelopedSignatureTransform would not work
- 1296 if there was a Canonicalization before it. Similar problems apply to transforms using the
- 1297 here() function. If such are to be supported, the use of Base64XML or EscapedXML MAY
- 1298 be required.
- 1299 c. Unchanged
- 1300 d. Unchanged
- 1301 i. Unchanged
- 1302 ii. Unchanged
- 1303 iii. Unchanged
- 1304 iv. Unchanged
- 1305 v. Unchanged (Note: the requirement imposed in 1.b of having the
- 1306 EnvelopedSignatureTransform as the first transform in the <ds:Transforms>
- 1307 list MUST be observed).
- 1308 2. Omitted

- 1309 3. Omitted
- 1310 b. After creating the `<ds:Reference>` due to the modified Basic Processing, make it available for
- 1311 the Basic Processing, as required in 3.3.1 Step 2.
- 1312 5. Add the returned `<ds:Reference>` as required in 3.3.1 Step 2 of Basic processing.

1313 3.5.9 Optional Input `<SignedReferences>`

1314 The `<SignedReferences>` element gives the client greater control over how the `<ds:Reference>`

1315 elements are formed. When this element is present, step 1 of Basic Processing (section 3.3.1) is

1316 overridden. Instead of there being a one-to-one correspondence between input documents and

1317 `<ds:Reference>` elements, now each `<SignedReference>` element controls the creation of a

1318 corresponding `<ds:Reference>`.

1319 Since each `<SignedReference>` refers to an input document, this allows multiple `<ds:Reference>`

1320 elements to be based on a single input document. Furthermore, the client can request additional

1321 transforms to be applied to each `<ds:Reference>`, and can set each `<ds:Reference>` element's Id

1322 or URI attribute. These aspects of the `<ds:Reference>` can only be set through the

1323 `<SignedReferences>` optional input; they cannot be set through the input documents, since they are

1324 aspects of the reference to the input document, not the input document itself.

1325 Each `<SignedReference>` element contains:

1326 `WhichDocument` [Required]

1327 Which input document this reference refers to (see the ID attribute in section 2.4.1).

1328 `RefId` [Optional]

1329 Sets the Id attribute of the corresponding `<ds:Reference>`.

1330 `RefURI` [Optional]

1331 If this attribute is present, the corresponding `<ds:Reference>` element's URI attribute is set to its

1332 value. If it is not present, the URI attribute is omitted in the corresponding `<ds:Reference>`

1333 `RefType` [Optional]

1334 overrides the `RefType` of `<dss:Document>`

1335 `<ds:Transforms>` [Optional]

1336 Requests the server to perform additional transforms on this reference.

1337 When the `<SignedReferences>` optional input is present, basic processing 3.3.1 step 1 is performed

1338 for each `<SignedReference>` overriding steps a., b., c. and d.:

1339 If the `<SignaturePlacement>` element is present the server processes it as follows:

1340 For each `<SignedReference>` in `<SignedReferences>`

1341 1. The server identifies the `<Document>` referenced as indicated by the `WhichDocument` attribute.

1342 2. If `RefURI` is present create an additional `<ds:Reference>` for the document in question by

1343 performing basic processing as in section 3.3.1 Step 1 amended as follows:

1344 1.

1345 a. Unchanged.

1346 b. Applies the transforms indicated in `<ds:Transforms>`. Afterwards, the server may apply

1347 any other transform it considers appropriate as per its policy and then generates a

1348 canonicalized octet string as required in step b. of basic Processing before hashing.

1349 c. Unchanged.

1350 d. The server forms a `<ds:Reference>` with the elements and attributes set as follows:

1351 i. Use this `RefURI` attribute from the `<SignedReference>` if present instead of

1352 `RefURI` from `<dss:Document>` in step i. of Basic Processing.

- 1353 The Id attribute is set to the <SignedReference> element's RefId attribute. If the
 1354 <SignedReference> has no RefId attribute, the <ds:Reference> element's
 1355 Id attribute is omitted.
- 1356 ii. Unchanged.
- 1357 iii. Unchanged.
- 1358 iv. Unchanged.
- 1359 v. The <ds:Transforms> used here will have to be added to <ds:Transforms> of
 1360 step v. of basic processing so that this element describes the sequence of transforms
 1361 applied by the server and describing the effective transform as a reproducible
 1362 procedure from parsing until hash.
- 1363 2. Add the returned <ds:Reference> as required in 3.3.1 Step 2 of Basic processing.
- 1364 3. If RefURI is not present perform basic processing for the input document not creating an additional
 1365 <ds:Reference> amending Step 1 as follows:
- 1366 1.
- 1367 a. Unchanged.
- 1368 b. Applies the transforms indicated in <ds:Transforms>. Afterwards, the server may apply
 1369 any other transform it considers as appropriate as per its policy and then generates
 1370 generating a canonicalized octet string as required in step b. of basic Processing before
 1371 hashing.
- 1372 c. Unchanged.
- 1373 d. The server forms a <ds:Reference> with the elements and attributes set as follows:
- 1374 i. Perform step i. of Basic Processing and the Id attribute is set to the
 1375 <SignedReference> element's RefId attribute. If the <SignedReference> has
 1376 no RefId attribute, the <ds:Reference> element's Id attribute is omitted.
- 1377 ii. Unchanged
- 1378 iii. Unchanged
- 1379 iv. Unchanged
- 1380 v. The <ds:Transforms> used here will have to be added to <ds:Transforms> of
 1381 step v. of basic processing so that this element describes the sequence of transforms
 1382 applied by the server and describing the effective transform as a reproducible
 1383 procedure from parsing until hash.
- 1384 4. The server continues with processing as specified in section 3.3.1 for the rest of the documents.

```

1385 <xs:element name="SignedReferences">
1386   <xs:complexType>
1387     <xs:sequence>
1388       <xs:element ref="dss:SignedReference"
1389         maxOccurs="unbounded" />
1390     </xs:sequence>
1391   </xs:complexType>
1392 </xs:element>
1393
1394 <xs:element name="SignedReference">
1395   <xs:complexType>
1396     <xs:sequence>
1397       <xs:element ref="ds:Transforms" minOccurs="0" />
1398     </xs:sequence>
1399     <xs:attribute name="WhichDocument" type="xs:IDREF" use="required" />
1400     <xs:attribute name="RefURI" type="xs:anyURI" use="optional" />
1401     <xs:attribute name="RefId" type="xs:string" use="optional" />
1402   </xs:complexType>
1403 </xs:element>

```

1404 4 The DSS Verifying Protocol

1405 4.1 Element <VerifyRequest>

1406 The <VerifyRequest> inherits from <RequestBaseType>. This element is sent by the client to verify
1407 a signature or timestamp on some input documents. It contains the following additional elements:

1408 <SignatureObject> [Optional]

1409 This element contains a signature or timestamp, or else contains a <SignaturePtr> that points to
1410 an XML signature in one of the input documents. If this element is omitted, there must be only a single
1411 <InputDocument> which the server will search to find the to-be-verified signature(s). Either a
1412 <SignaturePtr> or a single <InputDocument> and no <SignatureObject> MUST be used
1413 whenever the to-be-verified signature is an XML signature which uses an Enveloped Signature
1414 Transform; otherwise the server would have difficulty locating the signature and applying the
1415 Enveloped Signature Transform.

```
1416 <xs:element name="VerifyRequest">  
1417   <xs:complexType>  
1418     <xs:complexContent>  
1419       <xs:extension base="dss:RequestBaseType">  
1420         <xs:sequence>  
1421           <xs:element ref="dss:SignatureObject" minOccurs="0" />  
1422         </xs:sequence>  
1423       </xs:extension>  
1424     </xs:complexContent>  
1425   </xs:complexType>  
1426 </xs:element>
```

1427 4.2 Element <VerifyResponse>

1428 The <VerifyResponse> inherits from <ResponseBaseType>. This element defines no additional
1429 attributes and elements.

```
1430 <xs:element name="VerifyResponse" type="dss:ResponseBaseType" />
```

1431 4.3 Basic Processing for XML Signatures

1432 A DSS server that verifies XML signatures SHOULD perform the following steps, upon receiving a
1433 <VerifyRequest>. These steps may be changed or overridden by the optional inputs, or by the profile
1434 or policy the server is operating under. For more details on multi-signature verification, see section 4.3.1.

- 1435 1. The server retrieves one or more <ds:Signature> objects, as follows: If the
1436 <SignatureObject> is present, the server retrieves either the <ds:Signature> that is a child
1437 element of the <SignatureObject> (see: Note at the end of this section), or those
1438 <ds:Signature> objects which are pointed to by the <SignaturePtr> in the
1439 <SignatureObject>.
 - 1440 a. If the <SignaturePtr> points to an input document but not a specific element in that document,
1441 the pointed-to input document must be a <Document> element containing XML either in an
1442 <Base64XML>, <EscapedXML> or <InlineXML> element.
1443 If the document is inside <Base64XML> or <EscapedXML> it is decoded and parsed as
1444 described in 3.3.1 Step 1.a or 3.3.3 Step 1a respectively.
1445 If the document is inside <InlineXML> the document is extracted using exclusive
1446 canonicalization. The <ds:Reference> corresponding to the document MUST have a
1447 chain of transforms (at least one ds:Transform inside ds:Transforms) that anticipates

1448 and reflects this. If this is not the case the server MUST throw an Error
1449 (urn:oasis:names:tc:dss:1.0:resultminor:inappropriate:signature).
1450 Note: Otherwise false negatives due to namespace conflicts may appear.

1451 b. If the <SignatureObject> is omitted, there MUST be only a single <Document> element.
1452 This case is handled as if a <SignaturePtr> pointing to the single <Document> was
1453 present: the server will search and find every <ds:Signature> element in this input
1454 document, and verify each <ds:Signature> according to the steps below.

1455 2. For each <ds:Reference> in the <ds:Signature>, the server finds the input document with
1456 matching RefURI and RefType values (omitted attributes match omitted attributes). If the
1457 <ds:Reference> uses a same-document URI, the XPointer should be evaluated against the input
1458 document the <ds:Signature> is contained within, or against the <ds:Signature> itself if it is
1459 contained within the <SignatureObject> element. The <SchemaRef> element or optional input
1460 <Schema> of the input document or <SignatureObject> will be used, if present, to identify ID
1461 attributes when evaluating the XPointer expression. If the <ds:Reference> uses an external URI
1462 and the corresponding input document is not present, the server will skip the <ds:Reference>, and
1463 later return a result code such as ReferencedDocumentNotPresent to indicate this. The RefURI
1464 MAY be omitted in at most one of the set of Input documents.

1465 a. If the input document is a <Document>, the server extracts and decodes as described in
1466 3.3.1 Step 1.a (or equivalent step in variants of the basic process as defined in 3.3.2 onwards
1467 depending of the form of the input document).

1468 b. If the input document is a <TransformedData>, the server MAY check that the
1469 <ds:Transforms> (if supplied) match between the <TransformedData> and the
1470 <ds:Reference> and then hashes the resultant data object according to
1471 <ds:DigestMethod>, and MUST check that the result matches <ds:DigestValue>.

1472 c. If the input document is a <DocumentHash>, the server MAY check that the
1473 <ds:Transforms>, <ds:DigestMethod> (if supplied) and <ds:DigestValue> elements
1474 match between the <DocumentHash> and the <ds:Reference>.

1475 d. If the combination of RefURI and RefType matches more than one input document all of
1476 them MUST be either a <TransformedData> or a <DocumentHash> otherwise a
1477 RequesterError is issued qualified by result minor of
1478 ReferencedDocumentNotPresent.

1479 Only one of them is allowed to have a WhichReference value that matches the order of the
1480 <ds:Reference> within the <ds:SignedInfo> in question otherwise a RequesterError
1481 is issued qualified by result minor of ReferencedDocumentNotPresent. Using this input
1482 document either variant b. or c. is applied respectively before continuing with step 3.

1483 3. The server shall verify the validity of the signature at a particular time (i.e. current time, assumed
1484 signing time or other time), depending on the server policy. This behaviour MAY be altered by using
1485 the optional input <UseVerificationTime> (see section 4.5.2).

1486 4. If the signature validates correctly, the server returns one of the first three <ResultMinor> codes
1487 listed in section 4.4, depending on the relationship of the signature to the input documents (not
1488 including the relationship of the signature to those XML elements that were resolved through XPointer
1489 evaluation; the client will have to inspect those relationships manually). If the signature fails to
1490 validate correctly, the server returns some other code; either one defined in section 4.4 of this
1491 specification, or one defined by some profile of this specification.

1492 Note: The extraction of the <ds:Signature> from the <SignatureObject> should be performed
1493 without namespace inheritance. If the signature <ds:Signature> does not use exclusive
1494 canonicalization for it's <ds:CanonicalizationMethod> there can appear problems caused by
1495 namespace declarations moved by gateways or protocol processors of outer protocol bindings that alter
1496 the signature object and cause false negatives on validation. Problems appearing due to different
1497 behavior of xml parsers in schema validating parsing vs. non-validating parsing like data type
1498 normalizations would have to be healed by canonicalization only as no transforms are available for

1499 ds:SignedInfo. As currently available specifications of canonicalization are not aware of schema data
1500 types a solution to heal these defects is currently not possible. Beware, these problems can already occur
1501 on parsing the whole request including protocol bindings like SOAP. Implementors are encouraged to
1502 make use of <dss:Base64XML> or <dss: EscapedXML> instead.

1503 **4.3.1 Multi-Signature Verification**

1504 If a client requests verification of an entire input document, either using a <SignaturePtr> without an
1505 <XPath> or a missing <SignaturePtr> (see section 4.3 step 1), then the server **MUST** determine
1506 whether the input document contains zero, one, or more than one <ds:Signature> elements. If zero,
1507 the server should return a <ResultMajor> code of RequesterError.

1508 If more than one <ds:Signature> elements are present, the server **MUST** either reject the request with
1509 a <ResultMajor> code of RequesterError and a <ResultMinor> code of NotSupported, or
1510 accept the request and try to verify all of the signatures.

1511 If the server accepts the request in the multi-signature case (or if only a single signature is present) and
1512 one of the signatures fails to verify, the server should return one of the error codes in section 4.4,
1513 reflecting the first error encountered.

1514 If all of the signatures verify correctly, the server should return the Success <ResultMajor> code and
1515 the following <ResultMinor> code:

1516 urn:oasis:names:tc:dss:1.0:resultminor:ValidMultiSignatures

1517 Note: These procedures only define procedures for handling of multiple signatures on
1518 one input document. The procedures for handling multiple signatures on multiple
1519 documents are not defined in this core specification, but however such procedures, along
1520 with any optional elements that may be required, may be defined in profiles of this
1521 specification.

1522 Only certain optional inputs and outputs are allowed when performing multi-signature verification. See
1523 section 4.6 for details.

1524 **4.3.2 Signature Timestamp verification procedure**

1525 The following sub-sections will describe the processing rules for verifying:

- 1526 - RFC 3161 timestamp tokens on CMS Signatures
- 1527 - XML timestamp tokens on XML Signatures
- 1528 - RFC 3161 timestamp tokens on XML Signatures

1529 This section describes signature timestamp processing when the timestamp is embedded in the incoming
1530 signature.

1531 Note: procedures for handling other forms of timestamp may be defined in profiles of the Core. In
1532 particular, the DSS AdES profile [**DSS-AdES-P**] defines procedures for handling timestamps against the
1533 document being signed, and the DSS Timestamp profile defines procedures for handling standalone
1534 timestamps.

1535 For a definition of the <Timestamp> element see section 5.1 Details of the XML timestamp token can be
1536 found in subsection 5.1.1.

1537 **4.3.2.1 Processing for RFC 3161 Timestamp tokens on CMS Signatures.**

1538 The present section describes the processing rules for verifying a CMS RFC3161 timestamp token
1539 passed in on a Verify call within the <SignatureObject> of the <VerifyRequest> element. In the
1540 CMS case, since the "signature timestamp" is embedded in the signature as an unsigned attribute, only
1541 the time stamped signature is required for verification processing. As such, no additional input is required.

1542 The processing by the server is broken down into the following steps:

- 1543 1. The signature timestamp is embedded in the incoming signature as an unsigned attribute whose
1544 object identifier is 1.2.840.11359.1.9.16.2.14. Extract and verify the timestamp token.
- 1545 2. Verify that the token's public verification certificate is authorized for time stamping by examining the
1546 Extended Key Usage field for the presence of the time stamping OID "1.3.6.1.5.5.7.3.8".
- 1547 3. Validate that the `TstInfo` structure has a valid layout as defined in **[RFC 3161]**.
- 1548 4. Extract the `MessageImprint` hash value and associated algorithm from the `TstInfo` structure
1549 which will be compared against the hash value derived in the next step.
- 1550 5. Recalculate the hash of the signature value field of the signature in which the timestamp is
1551 embedded.
- 1552 6. Compare the hash values from the two previous steps, and if they are equivalent, then this timestamp
1553 is valid for the signature that was time stamped.
- 1554 7. Verify that the public verification certificate conforms to all relevant aspects of the relying-party's
1555 policy including algorithm usage, policy OIDs, time accuracy tolerances, and the Nonce value.
- 1556 8. Set the `dss:Result` element as defined in this specification. Minor Error
1557 `urn:oasis:names:tc:dss:1.0:resultminor:valid:signature:InvalidSignatureTim`
1558 `estamp` MAY be used to indicate that the signature is valid but the timestamp against that signature
1559 is invalid.

1560 **4.3.2.2 Processing for XML timestamp tokens on XML signatures**

1561 The present section describes the processing rules for verifying and XML Signature timestamp token
1562 embedded within an XML signature using the incorporation mechanisms specified in XAdES (i.e., in the
1563 `<xades:XMLTimeStamp>` `<xades:SignatureTimeStamp>` element's child). This XML signature may
1564 be passed in on a Verify call within the `<SignatureObject>` or embedded within a `<Document>`'s
1565 child.

1566 The server shall verify the timestamp token performing the steps detailed below. If any one of them
1567 results in failure, then the timestamp token SHOULD be rejected.

- 1568 9. Extract the timestamp token embedded in the incoming signature as defined in 3.5.2.2.
- 1569 10. Verify that the verification key and algorithms used conforms to all relevant aspects of the applicable
1570 policy. Should this key come within a public certificate, verify that the certificate conforms to all
1571 relevant aspects of the applicable policy including algorithm usage, policy OIDs, and time accuracy
1572 tolerances.
- 1573 11. Verify that the aforementioned verification key is consistent with the
1574 `ds:SignedInfo/SignatureMethod/@Algorithm` attribute value.
- 1575 12. Verify the timestamp token signature in accordance with the rules defined in **[XMLDSIG]**.
- 1576 13. Verify that the `<ds:SignedInfo>` element contains at least two `<ds:Reference>` elements.
- 1577 14. Verify that one of the `<ds:Reference>` elements has its `Type` attribute set to
1578 "urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken". Take this one and proceed as
1579 indicated below:
- 1580 a. Retrieve the referenced data object. Verify that it references a `<ds:Object>` element, which
1581 in turn envelopes a `<TSTInfo>` element.
 - 1582 b. Verify that the `<TSTInfo>` element has a valid layout as per the present specification.
 - 1583 c. Extract the digest value and associated algorithm from its `<ds:DigestValue>` and
1584 `<ds:DigestMethod>` elements respectively.
 - 1585 d. Recalculate the digest of the retrieved data object as specified by **[XMLDSIG]** with the digest
1586 algorithm indicated in `<ds:DigestMethod>`, and compare this result with the contents of
1587 `<ds:DigestValue>`.
- 1588 15. Take each of the other `<ds:Reference>` elements and for each validate the hash as specified in
1589 **[XMLDSIG]**.

- 1590 16. Check that for one of the `<ds:Reference>` elements the retrieved data object is actually the
1591 `<ds:SignatureValue>` element and that it contains its digest after canonicalization.
- 1592 17. Set the `<dss:Result>` element as appropriate. Minor Error
1593 `urn:oasis:names:tc:dss:1.0:resultminor:valid:signature:InvalidSignatureTim`
1594 `estamp` MAY be used to indicate that the signature is valid but the timestamp against that signature
1595 is invalid.

1596 4.3.2.3 Processing for RFC 3161 timestamp tokens on XML Signatures

1597 The present section describes the processing rules for verifying an RFC 3161 timestamp token
1598 embedded within an XML signature as an unsigned property. This XML signature may be passed in on a
1599 `Verify` call within the `<SignatureObject>` or embedded within a `<Document>`'s child.

1600 The server shall verify the timestamp token performing the steps detailed below. If any one of them
1601 results in failure, then the timestamp token SHOULD be rejected.

- 1602 1. Extract the timestamp token embedded in the incoming signature as defined in 3.5.2.3.
- 1603 2. Verify that the token's public verification certificate is authorized for time stamping by examining the
1604 Extended Key Usage field for the presence of the time stamping OID "1.3.6.1.5.5.7.3.8".
- 1605 3. Process the signature timestamp as defined in [XAdES] Annex G.2.2.16.1.3.
- 1606 4. Verify that the public verification certificate conforms to all relevant aspects of the relying-party's
1607 policy including algorithm usage, policy OIDs, time accuracy tolerances, and the Nonce value.
- 1608 5. Set the `dss:Result` element as appropriate.
1609 `urn:oasis:names:tc:dss:1.0:resultminor:valid:signature:InvalidSignatureTim`
1610 `estamp` MAY be used to indicate that the signature is valid but the timestamp against that signature
1611 is invalid.

1612 4.4 Basic Processing for CMS Signatures

1613 A DSS server that verifies CMS signatures SHOULD perform the following steps, upon receiving a
1614 `<VerifyRequest>`. These steps may be changed or overridden by the optional inputs, or by the profile
1615 or policy the server is operating under.

- 1616 1. The server retrieves the CMS signature by decoding the `<Base64Signature>` child of
1617 `<SignatureObject>`.
- 1618 2. The server retrieves the input data. If the CMS signature is detached, there must be a single input
1619 document: i.e. a single `<Document>` or `<DocumentHash>` element. Otherwise, if the CMS signature
1620 is enveloping, it contains its own input data and there MUST NOT be any input documents present.
- 1621 3. The CMS signature and input data are verified in the conventional way (see [RFC 3852] for details).
- 1622 4. If the signature validates correctly, the server returns the first `<ResultMinor>` code listed in section
1623 4.4. If the signature fails to validate correctly, the server returns some other code; either one defined
1624 in section 4.4 of this specification, or one defined by some profile of this specification.

1625 4.5 Optional Inputs and Outputs

1626 This section defines some optional inputs and outputs that profiles of the DSS verifying protocol might
1627 find useful. Section 2.8 defines some common optional inputs that can also be used with the verifying
1628 protocol. Profiles of the verifying protocol can define their own optional inputs and outputs, as well.
1629 General handling of optional inputs and outputs is discussed in section 2.7.

1630 4.5.1 Optional Input `<VerifyManifests>` and Output `<VerifyManifestResults>`

1631 The presence of this element instructs the server to validate manifests in an XML signature.

1632 On encountering such a document in step 2 of basic processing, the server shall repeat step 2 for all the
1633 `<ds:Reference>` elements within the manifest. In accordance with [XMLDSIG] section 5.1, DSS

1634 Manifest validation does not affect a signature's core validation. The results of verifying individual
1635 <ds:Reference>'s within a <ds:Manifest> are returned in the <dss:VerifyManifestResults>
1636 optional output.

1637 For example, a client supplies the optional input <VerifyManifests>, then the returned
1638 <ResultMinor> is

1639 urn:oasis:names:tc:dss:1.0:resultminor:valid:hasManifestResults if XMLSig core
1640 validation succeeds and the optional output <VerifyManifestResults> is returned indicating the
1641 status of the manifest reference verification. In case of a negative XMLSig core validation no attempt is
1642 made to verify manifests.

1643 The <VerifyManifests> optional input is allowed in multi-signature verification. The
1644 <VerifyManifestResults> is comprised of one or more <ManifestResult>s that contain the
1645 following:

1646 <ReferenceXPath> [Required]

1647 Identifies the manifest reference, in the XML signature, to which this result pertains.

1648 <Status> [Required]

1649 Indicates the manifest validation result. It takes one of the values
1650 urn:oasis:names:tc:dss:1.0:manifeststatus:Valid or urn:oasis:names:tc:dss:1.0:manifeststatus:Invalid.

```
1651 <xs:element name="VerifyManifestResults"  
1652 type="dss:VerifyManifestResultsType" />  
1653  
1654 <xs:complexType name="VerifyManifestResultsType">  
1655 <xs:sequence>  
1656 <xs:element ref="dss:ManifestResult" maxOccurs="unbounded" />  
1657 </xs:sequence>  
1658 </xs:complexType>  
1659  
1660 <xs:element name="ManifestResult">  
1661 <xs:complexType>  
1662 <xs:sequence>  
1663 <xs:element name="ReferenceXPath" type="xs:string" />  
1664 <xs:element name="Status" type="xs:anyURI" />  
1665 </xs:sequence>  
1666 </xs:complexType>  
1667 </xs:element>
```

1668 4.5.2 Optional Input <UseVerificationTime>

1669 This element instructs the server to attempt to determine the signature's validity at the specified time,
1670 instead of a time determined by the server policy.

1671 Note: In order to perform the verification of the signature at a certain time, the server MUST obtain the
1672 information necessary to carry out this verification (e.g. CA certificates, CRLs) applicable at that time.

1673 <CurrentTime> [Optional]

1674 Instructs the server to use its current time (normally the time associated with the server-side request
1675 processing).

1676 <SpecificTime> [Optional]

1677 Allows the client to manage manually the time instant used in the verification process. It SHOULD be
1678 expressed as UTC time (Coordinated Universal Time) to reduce confusion with the local time zone
1679 use.

1680 Profiles MAY define new child elements associated to other different behaviors.

```
1681 <xs:element name="UseVerificationTime" />  
1682 <xs:complexType name="UseVerificationTimeType">  
1683 <xs:choice>  
1684 <xs:element name="CurrentTime" />
```

1685
1686
1687
1688

```
<xs:element name="SpecificTime" type="xs:dateTime" />  
<xs:any namespace="##other" />  
</xs:choice>  
</xs:complexType>
```

1689 If the verification time is a significant period in the past the server MAY need to take specific steps for this,
1690 and MAY need to ensure that any cryptographic weaknesses over the period do not affect the validation.
1691 This optional input is allowed in multi-signature verification.

1692 **4.5.3 Optional Input/Output <ReturnVerificationTimeInfo> /** 1693 **<VerificationTimeInfo>**

1694 This element allows the client to obtain the time instant used by the server to validate the signature.

1695

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

1696 Optionally, in addition to the verification time, the server MAY include in the <VerificationTimeInfo>
1697 response any other relevant time instants that may have been used when determining the verification
1698 time or that may be useful for its qualification.

1699 <VerificationTime> [Required]

1700 The time instant used by the server when verifying the signature. It SHOULD be expressed as UTC
1701 time (Coordinated Universal Time) to reduce confusion with the local time zone use.

1702 <AdditionalTimeInfo> [Optional]

1703 Any other time instant(s) relevant in the context of the verification time determination.

1704 The Type attribute qualifies the kind of time information included in the response. The Ref attribute allows
1705 to establish references to the source of the time information, and SHOULD be used when there is a need
1706 to disambiguate several <AdditionalTimeInfo> elements with the same Type attribute.

1707 This specification defines the following base types, whose values MUST be of type xs:dateTime and
1708 SHOULD be expressed as UTC time (Coordinated Universal Time). Profiles MAY include and define new
1709 values for the Type attribute.

1710 urn:oasis:names:tc:dss:1.0:additionaltimeinfo:signatureTimestamp

1711 The time carried inside a timestamp applied over the signature value.

1712 urn:oasis:names:tc:dss:1.0:additionaltimeinfo:signatureTimemark

1713 The time instant associated to the signature stored in a secure record in the server.

1714 urn:oasis:names:tc:dss:1.0:additionaltimeinfo:signedObjectTimestamp

1715 The time carried inside a timestamp applied over a signed object.

1716 Note that XML Signatures can be produced over multiple objects (via multiple ds:Reference
1717 elements), and therefore it's possible to have multiple timestamps, each one applied over each object.
1718 In this case, the Ref attribute MUST include the value of the Id attribute of the ds:Reference element.

1719 urn:oasis:names:tc:dss:1.0:additionaltimeinfo:claimedSigningTime

1720 The time claimed by the signer to be the signature creation time.

1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732

```
<xs:element name="AdditionalTimeInfo" type="dss:AdditionalTimeInfoType" />  
<xs:complexType name="AdditionalTimeInfoType">  
<xs:simpleContent>  
<xs:extension base="xs:dateTime">  
<xs:attribute name="Type" type="xs:anyURI" use="required" />  
<xs:attribute name="Ref" type="xs:string" use="optional" />  
</xs:extension>  
</xs:simpleContent>  
</xs:complexType>  
<xs:element name="VerificationTimeInfo"  
type="dss:VerificationTimeInfoType" />  
<xs:complexType name="VerificationTimeInfoType">
```

```

1733 <xs:sequence>
1734   <xs:element name="VerificationTime" type="xs:dateTime" />
1735   <xs:element ref="dss:AdditionalTimeInfo" minOccurs="0"
1736             maxOccurs="unbounded" />
1737 </xs:sequence>
1738 </xs:complexType>

```

1739 In the case of multi-signature verification, it's a matter of server policy as to whether this element is
1740 supported.

1741 This optional input is not allowed in multi-signature verification.

1742 4.5.4 Optional Input <AdditionalKeyInfo>

1743 This element provides the server with additional data (such as certificates and CRLs) which it can use to
1744 validate the signature.

1745 This optional input is not allowed in multi-signature verification.

```

1746 <xs:element name="AdditionalKeyInfo">
1747   <xs:complexType>
1748     <xs:sequence>
1749       <xs:element ref="ds:KeyInfo" />
1750     </xs:sequence>
1751   </xs:complexType>
1752 </xs:element>

```

1753 4.5.5 Optional Input <ReturnProcessingDetails> and Output 1754 <ProcessingDetails>

1755 The presence of the <ReturnProcessingDetails> optional input instructs the server to return a
1756 <ProcessingDetails> output.

1757 These options are not allowed in multi-signature verification.

```

1758 <xs:element name="ReturnProcessingDetails" />

```

1759 The <ProcessingDetails> optional output elaborates on what signature verification steps succeeded
1760 or failed. It may contain the following child elements:

1761 <ValidDetail> [Any Number]

1762 A verification detail that was evaluated and found to be valid.

1763 <IndeterminateDetail> [Any Number]

1764 A verification detail that could not be evaluated or was evaluated and returned an indeterminate result.

1765 <InvalidDetail> [Any Number]

1766 A verification detail that was evaluated and found to be invalid.

```

1767 <xs:element name="ProcessingDetails">
1768   <xs:complexType>
1769     <xs:sequence>
1770       <xs:element name="ValidDetail" type="dss:DetailType"
1771             minOccurs="0" maxOccurs="unbounded" />
1772       <xs:element name="IndeterminateDetail"
1773             type="dss:DetailType"
1774             minOccurs="0" maxOccurs="unbounded" />
1775       <xs:element name="InvalidDetail" type="xs:dss:DetailType"
1776             minOccurs="0" maxOccurs="unbounded" />
1777     </xs:sequence>
1778   </xs:complexType>
1779 </xs:element>

```

1780 Each detail element is of type `dss:DetailType`. A `dss:DetailType` contains the following child
 1781 elements and attributes:

1782 `Type` [Required]

1783 A URI which identifies the detail. It may be a value defined by this specification, or a value defined by
 1784 some other specification. For the values defined by this specification, see below.

1785 Multiple detail elements of the same `Type` may appear in a single `<ProcessingDetails>`. For
 1786 example, when a signature contains a certificate chain that certifies the signing key, there may be details
 1787 of the same `Type` present for each certificate in the chain, describing how each certificate was
 1788 processed.

1789 `<Code>` [Optional]

1790 A URI which more precisely specifies why this detail is valid, invalid, or indeterminate. It must be a
 1791 value defined by some other specification, since this specification defines no values for this element.

1792 `<Message>` [Optional]

1793 A human-readable message which MAY be logged, used for debugging, etc.

```

1794 <xs:complexType name="DetailType">
1795   <xs:sequence>
1796     <xs:element name="Code" type="xs:anyURI" minOccurs="0" />
1797     <xs:element name="Message" type="dss:InternationalStringType"
1798       minOccurs="0" />
1799     <xs:any namespace="##other" processContents="lax" minOccurs="0"
1800       maxOccurs="unbounded" />
1801   </xs:sequence>
1802   <xs:attribute name="Type" type="xs:anyURI" use="required" />
1803 </xs:complexType>
  
```

1804 The values for the `Type` attribute defined by this specification are the following:

1805 `urn:oasis:names:tc:dss:1.0:detail:IssuerTrust`

1806 Whether the issuer of trust information for the signing key (or one of the certifying keys) is considered
 1807 to be trustworthy.

1808 `urn:oasis:names:tc:dss:1.0:detail:RevocationStatus`

1809 Whether the trust information for the signing key (or one of the certifying keys) is revoked.

1810 `urn:oasis:names:tc:dss:1.0:detail:ValidityInterval`

1811 Whether the trust information for the signing key (or one of the certifying keys) is within its validity
 1812 interval.

1813 `urn:oasis:names:tc:dss:1.0:detail:Signature`

1814 Whether the document signature (or one of the certifying signatures) verifies correctly.

1815 `urn:oasis:names:tc:dss:1.0:detail:ManifestReference`

1816 Whether a manifest reference in the XML signature verified correctly.

1817 **4.5.6 Optional Input `<ReturnSigningTimeInfo>` and Output**
 1818 **`<SigningTimeInfo>`**

1819 This element allows the client to obtain the time instant associated to the signature creation.
 1820 Note: The signing time may be derived, for example, from a claimed signing time signed signature
 1821 attribute.

```

1822 <xs:element name="ReturnSigningTimeInfo" />
  
```

1823 Sometimes, depending on the applicable server policy, this signing time needs to be qualified, in order to
 1824 avoid unacceptable measurement errors or false claims, using time boundaries associated to trustworthy
 1825 time values (based on timestamps or time-marks created using trusted time sources). In this case, the

1826 server MAY include these values in the <LowerBoundary> and <UpperBoundary> elements,
1827 respectively.

1828 Criteria for determining when a time instant can be considered trustworthy and for determining the
1829 maximum acceptable delays between the signing time and their boundaries (if any) is outside the scope
1830 of this specification.

1831 When there's no way for the server to determine the signing time, the server MUST omit the
1832 <SigningTimeInfo> output.

1833 <SigningTime> [Required]

1834 The time value considered by the server to be the signature creation time.

1835 <SigningTimeBoundaries> [Optional]

1836 The trusted time values considered as lower and upper limits for the signing time. If this element is
1837 present, at least one of the <LowerBoundary> and <UpperBoundary> elements MUST be
1838 present.

```
1839 <xs:element name="SigningTimeInfo" type="dss:SigningTimeInfoType"/>
1840 <xs:complexType name="SigningTimeInfoType">
1841   <xs:sequence>
1842     <xs:element name="SigningTime" type="xs:dateTime"/>
1843     <xs:element name="SigningTimeBoundaries" minOccurs="0">
1844       <xs:complexType>
1845         <xs:sequence>
1846           <xs:element name="LowerBoundary" minOccurs="0"
1847             type="xs:dateTime"/>
1848           <xs:element name="UpperBoundary" minOccurs="0"
1849             type="xs:dateTime"/>
1850         </xs:sequence>
1851       </xs:complexType>
1852     </xs:element>
1853   </xs:sequence>
1854 </xs:complexType>
```

1855 This optional input is not allowed in multi-signature verification.

1856 4.5.7 Optional Input <ReturnSignerIdentity> and Output <SignerIdentity>

1857 The presence of the <ReturnSignerIdentity> optional input instructs the server to return a
1858 <SignerIdentity> output.

1859 This optional input and output are not allowed in multi-signature verification.

```
1860 <xs:element name="ReturnSignerIdentity"/>
```

1861 The <SignerIdentity> optional output contains an indication of who performed the signature.

```
1862 <xs:element name="SignerIdentity" type="saml:NameIdentifierType"/>
```

1863 4.5.8 Optional Input <ReturnUpdatedSignature> and Outputs 1864 <DocumentWithSignature>, <UpdatedSignature>

1865 The presence of the <ReturnUpdatedSignature> optional input instructs the server to return an
1866 <UpdatedSignature> output, containing a new or updated signature.

1867 The Type attribute on <ReturnUpdatedSignature>, if present, defines exactly what it means to
1868 "update" a signature. For example, the updated signature may be the original signature with some
1869 additional unsigned signature properties added to it (such as timestamps, counter-signatures, or
1870 additional information for use in verification), or the updated signature could be an entirely new signature
1871 calculated on the same input documents as the input signature. Profiles that use this optional input
1872 MUST define the allowed values and their semantics, and the default value, for the Type attribute (unless
1873 only a single type of updated signature is supported, in which case the Type attribute can be omitted).

1874 Multiple occurrences of this optional input can be present in a single verify request message. If multiple
1875 occurrences are present, each occurrence MUST have a different Type attribute. Each occurrence will
1876 generate a corresponding optional output. These optional outputs SHALL be distinguishable based on
1877 their Type attribute, which will match each output with an input.

1878 <UpdatedSignature>/<SignatureObject> [Optional]

1879 The resulting updated signature or timestamp or, in the case of a signature being enveloped in an
1880 output document, a pointer to the signature. This is used in steps 2. and 3. in the processing
1881 described below. These options are not allowed in multi-signature verification.

```
1882 <xs:element name="ReturnUpdatedSignature">  
1883 <xs:complexType>  
1884 <xs:attribute name="Type" type="xs:anyURI" use="optional" />  
1885 </xs:complexType>  
1886 </xs:element>
```

1887 The <UpdatedSignature> optional output contains the returned signature.

```
1888 <xs:element name="UpdatedSignature" type="dss:UpdatedSignatureType" />
```

1889 The <UpdatedSignatureType> is as follows.

```
1890 <xs:complexType name="UpdatedSignatureType">  
1891 <xs:sequence>  
1892 <xs:element ref="dss:SignatureObject" />  
1893 </xs:sequence>  
1894 <xs:attribute name="Type" type="xs:anyURI" use="optional" />  
1895 </xs:complexType>
```

1896 A DSS server SHOULD perform the following steps, upon receiving a <ReturnUpdatedSignature>.
1897 These steps may be changed or overridden by a profile or policy the server is operating under. (e.g For
1898 PDF documents enveloping cms signatures)

- 1899 1. If the signature to be verified and updated appears within a <SignatureObject>'s
1900 <ds:Signature> (detached or enveloping) or <Base64Signature> then the
1901 <UpdatedSignature> optional output MUST contain the modified <SignatureObject> with the
1902 corresponding <ds:Signature> (detached or enveloping) or <Base64Signature> child
1903 containing the updated signature.
- 1904 2. If the signature to be verified and updated is enveloped, and if the <VerifyRequest> contains a
1905 <SignatureObject> with a <SignaturePtr> pointing to an <InputDocument> (<Base64XML>,
1906 <InlineXML>, <EscapedXML>) enveloping the signature then the server MUST produce the
1907 following TWO optional outputs, first a <DocumentWithSignature> optional output containing the
1908 document that envelops the updated signature, second an <UpdatedSignature> optional output
1909 containing a <SignatureObject> having a <SignaturePtr> element that MUST point to the
1910 former <DocumentWithSignature>.
- 1911 3. If there is no <SignatureObject> at all in the request then the server MUST produce only a
1912 <DocumentWithSignature> optional output containing the document with the updated signature.
1913 No <UpdatedSignature> element will be generated.

1914 As <DocumentWithSignature> appears in steps 2. and 3. of the processing above it is explained
1915 here again:

1916 The <DocumentWithSignature> optional output (for the schema refer to section 3.5.8) contains the
1917 input document with the given signature inserted.

1918 It has one child element:

1919 <Document> [Required]

1920 This returns the given document with a signature inserted in some fashion.

1921 The resulting document with the updated enveloped signature is placed in the optional output
1922 <DocumentWithSignature>. The server places the signature in the document identified using the
1923 <SignatureObject>/<SignaturePtr>'s WhichDocument attribute.
1924 This <Document> MUST include a same-documentRefURI attribute which references the data updated
1925 (e.g of the form RefURI).

1926 **4.5.9 Optional Input <ReturnTransformedDocument> and Output** 1927 **<TransformedDocument>**

1928 The <ReturnTransformedDocument> optional input instructs the server to return an input document to
1929 which the XML signature transforms specified by a particular <ds:Reference> have been applied. The
1930 <ds:Reference> is indicated by the zero-based WhichReference attribute (0 means the first
1931 <ds:Reference> in the signature, 1 means the second, and so on). Multiple occurrences of this
1932 optional input can be present in a single verify request message. Each occurrence will generate a
1933 corresponding optional output.

1934 These options are not allowed in multi-signature verification.

```
1935 <xs:element name="ReturnTransformedDocument" >  
1936 <xs:complexType>  
1937 <xs:attribute name="WhichReference" type="xs:integer"  
1938 use="required" />  
1939 </xs:complexType>  
1940 </xs:element>
```

1941 The <TransformedDocument> optional output contains a document corresponding to the specified
1942 <ds:Reference>, after all the transforms in the reference have been applied. In other words, the hash
1943 value of the returned document should equal the <ds:Reference> element's <ds:DigestValue>. To
1944 match outputs to inputs, each <TransformedDocument> will contain a WhichReference attribute
1945 which matches the corresponding optional input.

```
1946 <xs:element name="TransformedDocument" >  
1947 <xs:complexType>  
1948 <xs:sequence>  
1949 <xs:element ref="dss:Document" >  
1950 </xs:sequence>  
1951 </xs:complexType>  
1952 <xs:attribute name="WhichReference" type="xs:integer"  
1953 use="required" />  
1954 </xs:element>
```

1955 **4.5.10 Optional Input <ReturnTimestampedSignature> and Outputs** 1956 **<DocumentWithSignature>, <TimestampedSignature>**

1957 The <ReturnTimestampedSignature> element within a <VerifyRequest> message indicates that
1958 the client wishes the server to update the signature after its verification by embedding a signature
1959 timestamp token as an unauthenticated attribute (see "unauthAttrs" in section 9.1 [RFC 3852]) or
1960 *unsigned* property (see section 6.2.5 "The UnsignedSignatureProperties element" and section 7.3 "The
1961 SignatureTimeStamp element" [XAdES]) of the supplied signature.

1962 The timestamp token will be on the signature value in the case of CMS/PKCS7 signatures or the
1963 <ds:SignatureValue> element in the case of XML signatures.

1964 The Type attribute, if present, indicates what type of timestamp to apply. This document defines two
1965 values for it, namely:

- 1966 a. urn:ietf:rfc:3161 for generating a RFC 3161 timestamp token on the signature
- 1967 b. urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken, for generating a XML
1968 timestamp token as defined in section 5 of this document.

1969 Profiles that use this optional input MUST define the allowed values, and the default value, for the Type
1970 attribute (unless only a single type of timestamp is supported, in which case the Type attribute can be
1971 omitted).

1972 Below follows the schema definition for these elements.

```
1973 <xs:element name="ReturnTimestampedSignature"  
1974           type="dss:UpdateSignatureInstructionType"/>  
1975 <xs:element name="TimestampedSignature" type="dss:UpdatedSignatureType"/>  
1976  
1977 <xs:element name="UpdatedSignature" type="dss:UpdatedSignatureType"/>  
1978   <xs:complexType name="UpdatedSignatureType">  
1979     <xs:sequence>  
1980       <xs:element ref="dss:SignatureObject"/>  
1981     </xs:sequence>  
1982     <xs:attribute name="Type" type="xs:anyURI" use="optional"/>  
1983   </xs:complexType>
```

1984 A DSS server SHOULD perform the steps 1. - 3. as indicated in 4.5.8 upon receiving a
1985 <ReturnTimeStampedSignature> replacing <UpdatedSignature> by
1986 <TimestampedSignature>.

1987 Procedures for handling RFC 3161 and XML timestamps are as defined in 3.5.2.3 and 3.5.2.2.

1988 Note: Procedures for handling other forms of timestamp may be defined in profiles of the Core. In
1989 particular, the DSS XAdES profile **[DSS-XAdES-P]** defines procedures for handling timestamps against
1990 the document being signed, and the DSS Timestamp profile **[DSS-TS-P]** defines procedures for handling
1991 standalone timestamps.

5 DSS Core Elements

1992

1993 This section defines two XML elements that may be used in conjunction with the DSS core protocols.

5.1 Element <Timestamp>

1994

1995 This section defines an XML timestamp. A <Timestamp> contains some type of timestamp token, such
1996 as an RFC 3161 TimeStampToken [RFC 3161] or a <ds:Signature> (aka an "XML timestamp token")
1997 (see section 5.1.1). Profiles may introduce additional types of timestamp tokens. Standalone XML
1998 timestamps can be produced and verified using the timestamping profile of the DSS core protocols [XML-
1999 TSP].

2000 An XML timestamp may contain:

2001 <ds:Signature> [Optional]

2002 This is an enveloping XML signature, as defined in section 5.1.1.

2003 <RFC3161TimeStampToken> [Optional]

2004 This is a base64-encoded TimeStampToken as defined in [RFC3161].

```
2005 <xs:element name="Timestamp">  
2006   <xs:complexType>  
2007     <xs:choice>  
2008       <xs:element ref="ds:Signature"/>  
2009       <xs:element name="RFC3161TimeStampToken"  
2010         type="xs:base64Binary"/>  
2011       <xs:element name="Other" type="AnyType"/>  
2012     </xs:choice>  
2013   </xs:complexType>  
2014 </xs:element>
```

5.1.1 XML Timestamp Token

2015

2016 An XML timestamp token is similar to an RFC 3161 TimeStampToken, but is encoded as a <TstInfo>
2017 element (see section 5.1.2) inside an enveloping <ds:Signature>. This allows conventional XML
2018 signature implementations to validate the signature, though additional processing is still required to
2019 validate the timestamp properties (see section 4.3.2.2).

2020 The following text describes how the child elements of the <ds:Signature> MUST be used:

2021 <ds:KeyInfo> [Required]

2022 The <ds:KeyInfo> element SHALL identify the issuer of the timestamp and MAY be used to
2023 locate, retrieve and validate the timestamp token signature-verification key. The exact details of
2024 this element may be specified further in a profile.

2025 <ds:SignedInfo>/<ds:Reference> [Required]

2026 There MUST be a single <ds:Reference> element whose URI attribute references the
2027 <ds:Object> containing the enveloped <TstInfo> element, and whose Type attribute is equal
2028 to urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken.

2029 <ds:Object> [Required]

2030 A <TstInfo> element SHALL be contained in a <ds:Object> element.

2031 Additional <ds:Reference> elements MUST appear for data objects [XMLDSIG] being time-stamped.

2032 For details on further use of time-stamps, please refer to appropriate profiles.

2033 5.1.2 Element <TstInfo>

2034 A <TstInfo> element is included in an XML timestamp token as a <ds:Signature> /
2035 <ds:Object> child element. A <TstInfo> element has the following children:

2036 <SerialNumber> [Required]

2037 This element SHALL contain a serial number produced by the timestamp authority (TSA). It
2038 MUST be unique across all the tokens issued by a particular TSA.

2039 <CreationTime> [Required]

2040 The time at which the token was issued.

2041 <Policy> [Optional]

2042 This element SHALL identify the policy under which the token was issued. The TSA's policy
2043 SHOULD identify the fundamental source of its time.

2044 <ErrorBound> [Optional]

2045 The TSA's estimate of the maximum error in its local clock.

2046 <Ordered> [Default="false"]

2047 This element SHALL indicate whether or not timestamps issued by this TSA, under this policy,
2048 are strictly ordered according to the value of the CreationTime element value.

2049 TSA [Optional]

2050 The name of the TSA.

```
2051 <xs:element name="TstInfo">  
2052 <xs:complexType>  
2053 <xs:sequence>  
2054 <xs:element name="SerialNumber" type="xs:integer"/>  
2055 <xs:element name="CreationTime" type="xs:dateTime"/>  
2056 <xs:element name="Policy" type="xs:anyURI" minOccurs="0"/>  
2057 <xs:element name="ErrorBound" type="xs:duration"  
2058 minOccurs="0"/>  
2059 <xs:element name="Ordered" type="xs:boolean"  
2060 default="false" minOccurs="0"/>  
2061 <xs:element name="TSA" type="saml:NameIdentifierType"  
2062 minOccurs="0"/>  
2063 </xs:sequence>  
2064 </xs:complexType>  
2065 </xs:element>
```

2066 5.2 Element <RequesterIdentity>

2067 This section contains the definition of an XML Requester Identity element. This element can be used as
2068 a signature property in an XML signature to identify the client who requested the signature.

2069 This element has the following children:

2070 Name [Required]

2071 The name or role of the requester who requested the signature be performed.

2072 SupportingInfo [Optional]

2073 Information supporting the name (such as a SAML Assertion [SAMLCore1.1], Liberty Alliance
2074 Authentication Context, or X.509 Certificate).

2075 The following schema fragment defines the <RequesterIdentity> element:

```
2076 <xs:element name="RequesterIdentity">  
2077 <xs:complexType>  
2078 <xs:sequence>
```

2079
2080
2081
2082
2083
2084

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

2085

6 DSS Core Bindings

2086 Mappings from DSS messages into standard communications protocols are called DSS *bindings*.
2087 *Transport bindings* specify how DSS messages are encoded and carried over some lower-level transport
2088 protocol. *Security bindings* specify how confidentiality, authentication, and integrity can be achieved for
2089 DSS messages in the context of some transport binding.
2090 Below we specify an initial set of bindings for DSS. Future bindings may be introduced by the OASIS
2091 DSS TC or by other parties.

2092

6.1 HTTP POST Transport Binding

2093 In this binding, the DSS request/response exchange occurs within an HTTP POST exchange [RFC
2094 2616]. The following rules apply to the HTTP request:
2095 The client may send an HTTP/1.0 or HTTP/1.1 request.
2096 The Request URI may be used to indicate a particular service endpoint.
2097 The `Content-Type` header MUST be set to "application/xml".
2098 The `Content-Length` header MUST be present and correct.
2099 The DSS request message MUST be sent in the body of the HTTP Request.
2100 The following rules apply to the HTTP Response:
2101 The `Content-Type` header MUST be set to "text/xml".
2102 The `Content-Length` header MUST be present and correct.
2103 The DSS response message MUST be sent in the body of the HTTP Response.
2104 The HTTP status code MUST be set to 200 if a DSS response message is returned. Otherwise, the
2105 status code can be set to 3xx to indicate a redirection, 4xx to indicate a low-level client error (such as a
2106 malformed request), or 5xx to indicate a low-level server error.

2107

6.2 SOAP 1.2 Transport Binding

2108 In this binding, the DSS request/response exchange occurs using the SOAP 1.2 message protocol
2109 [SOAP]. The following rules apply to the SOAP request:
2110 A single DSS `<SignRequest>` or `<VerifyRequest>` element will be transmitted within the body of the
2111 SOAP message.
2112 The client MUST NOT include any additional XML elements in the SOAP body.
2113 The UTF-8 character encoding must be used for the SOAP message.
2114 Arbitrary SOAP headers may be present.
2115 The following rules apply to the SOAP response:
2116 The server MUST return either a single DSS `<SignResponse>` or `<VerifyResponse>` element within
2117 the body of the SOAP message, or a SOAP fault code.
2118 The server MUST NOT include any additional XML elements in the SOAP body.
2119 If a DSS server cannot parse a DSS request, or there is some error with the SOAP envelope, the server
2120 MUST return a SOAP fault code. Otherwise, a DSS result code should be used to signal errors.
2121 The UTF-8 character encoding must be used for the SOAP message.
2122 Arbitrary SOAP headers may be present.
2123 On receiving a DSS response in a SOAP message, the client MUST NOT send a fault code to the DSS
2124 server.

2125 **6.2.1 SOAP Attachment Feature and Element <AttachmentReference>**

2126 Applications MAY support SOAP 1.2 attachment feature **[SOAPAtt]** to transmit documents in the context
2127 of a <SignRequest> or a <VerifyRequest> and can take advantage of
2128 <Document>/<AttachmentReference>.

2129 AttRefURI

2130 SOAP 1.2 attachment feature **[SOAPAtt]** states that any secondary part ("attachment") can be
2131 referenced by a URI of any URI scheme.

2132 AttRefURI refers to such a secondary part ("attachment") and MUST resolve within the
2133 compound SOAP message. The default encapsulation mechanism is MIME as specified in the
2134 WS-I Attachments Profile **[WS-I-Att]** (cf. swaRef, http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html#Referencing_Attachments_from_the_SOAP_Envelope).

2136 MimeType [Optional]

2137 Declares the MIME type of the referred secondary part of this SOAP compound message.

2138 Note: If MIME is used as encapsulation mechanism, the MIME content-type is available via a
2139 MIME header. However, the MIME headers may not be available to implementations and the
2140 SOAP 1.2 attachment feature is not restricted to MIME. Further the MIME header is not secured
2141 by the AttachmentReference's DigestValue, which is calculated over the binary attachment
2142 data (not including the MIME headers).

2143 <ds:DigestMethod> [Optional Sequence]

2144 <ds:DigestValue>

2145 These optional elements can be used to ensure the integrity of the attachment data.

2146 If these elements are supplied the server SHOULD compute a message digest using the
2147 algorithm given in <ds:DigestMethod> over the binary data in the octet stream and compare it
2148 against the supplied <ds:DigestValue>.

2149 If the comparison fails then a RequesterError qualified by a GeneralError and an
2150 appropriate message containing the AttRefURI is returned.

2151 Note: The attachments digest value(s) can be included in the primary SOAP part to allow the
2152 entire request (including secondary parts) to be secured by WSS. However, the MIME headers
2153 are not covered by the digest value and therefore can be included into the
2154 dss:AttachmentReference (which is relevant for the processing of dss:IncludeObject
2155 referring to an dss:AttachmentReference).

2156 The digest value may be computed while the data is read from the attachment. After the last byte
2157 being read from the attachment the server compares the calculated digest value against the
2158 supplied <ds:DigestValue>.

```
2159 <xs:element name="AttachmentReference" type="dss:AttachmentReferenceType" />
2160 <xs:complexType name="AttachmentReferenceType">
2161 <xs:sequence minOccurs="0">
2162 <xs:element ref="ds:DigestMethod" />
2163 <xs:element ref="ds:DigestValue" />
2164 </xs:sequence>
2165 <xs:attribute name="AttRefURI" type="xs:anyURI" />
2166 <xs:attribute name="MimeType" type="xs:string" use="optional" />
2167 </xs:complexType>
```

2168 **6.2.1.1 Signing Protocol, Processing for XML Signatures, Process Variant for** 2169 **<AttachmentReference>**

2170 In the case of an input document which contains <AttachmentReference> the server retrieves the
2171 MIME type from the MimeType attribute (if present) otherwise from the content-type MIME header of the

2172 attachment referred by `AttRefURI`. If the `MimeType` attribute diverges from the attachment's MIME
2173 header content-type, an implementation MAY either ignore the MIME header's content-type or issue a
2174 `RequesterError` qualified by a `GeneralError` and an appropriate message containing the
2175 `AttRefURI`.

2176 IF the MIME type indicates that it contains XML continue with processing as in section 3.3.1 and Step 1 a
2177 is replaced with the following:

2178 1.

2179 a. The server retrieves the data from the attachment referred by `AttRefURI` as an octet stream. This
2180 data MUST be a well formed XML Document as defined in [XML] section 2.1. If the `RefURI` attribute
2181 references within the same input document then the server parses the octet stream to `NodeSetData`
2182 (see [XMLDSIG] section 4.3.3.3) before proceeding to the next step.

2183 ELSE continue with processing as in section 3.3.4 and Step 1 a is replaced with the following:

2184 1.

2185 a. The server retrieves the data from the attachment referred by `AttRefURI` as an octet stream.

2186 Note: In the first case `attachmentReference` is always treated like `Base64XML` in the latter like
2187 `Base64Data` for further processing. (E.g. In the case of `dss:IncludeObject`, the `MimeType` attribute
2188 is copied from `dss:AttachmentReference` to `ds:Object`.)

2189 **6.2.1.2 Verifying Protocol, Processing for XML Signatures, Process Variant for** 2190 **<AttachmentReference>**

2191 Perform section 4.3 Basic Processing for XML Signatures amending step 2 2.a as follows:

2192 2.

2193 a. If the input document is a `<Document>`, the server extracts and decodes as described in 3.3.1 Step 1
2194 a (or equivalent step in variants of the basic process as defined in 3.3.2 onwards depending of the form of
2195 the input document) or in the case of `<AttachmentReference>` as described in section 6.2.1.1.

2196 **6.2.1.3 Signing Protocol, Basic Processing for CMS Signatures, Process Variant** 2197 **for <AttachmentReference>**

2198 Perform section 3.4 Basic Processing for CMS Signatures adding the following variant 1. d' after 1.d and
2199 before 1.e:

2200 1.

2201 d'. If the `<Document>` contains `<AttachmentReference>`, the server retrieves the data from the
2202 attachment referred by `AttRefURI` as an octet stream.

2203 **6.2.1.4 Verifying Protocol, Basic Processing for CMS Signatures, Process Variant** 2204 **for <AttachmentReference>**

2205 Perform section 4.4 Basic Processing for CMS Signatures amending step 2 as follows:

2206

2207 2. The server retrieves the input data. (In the case of `<AttachmentReference>` this is done as in
2208 section 6.2.1.3 step 1. d'. If the CMS signature is detached, there must be a single input document: i.e. a
2209 single `<Document>` or `<DocumentHash>` element. Otherwise, if the CMS signature is enveloping, it
2210 contains its own input data and there MUST NOT be any input documents present.

2211 **6.3 TLS Security Bindings**

2212 TLS [RFC 2246] is a session-security protocol that can provide confidentiality, authentication, and
2213 integrity to the HTTP POST transport binding, the SOAP 1.2 transport binding, or others. TLS supports a

2214 variety of authentication methods, so we define several security bindings below. All of these bindings
2215 inherit the following rules:
2216 TLS 1.0 MUST be supported. SSL 3.0 MAY be supported. Future versions of TLS MAY be supported.
2217 RSA ciphersuites MUST be supported. Diffie-Hellman and DSS ciphersuites MAY be supported.
2218 TripleDES ciphersuites MUST be supported. AES ciphersuites SHOULD be supported. Other
2219 ciphersuites MAY be supported, except for weak ciphersuites intended to meet export restrictions, which
2220 SHOULD NOT be supported.

2221 **6.3.1 TLS X.509 Server Authentication**

2222 The following ciphersuites defined in [RFC 2246] and [RFC 3268] are supported. The server MUST
2223 authenticate itself with an X.509 certificate chain [RFC 3280]. The server MUST NOT request client
2224 authentication.

2225 MUST:

2226 TLS_RSA_WITH_3DES_EDE_CBC_SHA

2227 SHOULD:

2228 TLS_RSA_WITH_AES_128_CBC_SHA

2229 TLS_RSA_WITH_AES_256_CBC_SHA

2230 **6.3.2 TLS X.509 Mutual Authentication**

2231 The same ciphersuites mentioned in section 6.2.1 are supported. The server MUST authenticate itself
2232 with an X.509 certificate chain, and MUST request client authentication. The client MUST authenticate
2233 itself with an X.509 certificate chain.

2234 **6.3.3 TLS SRP Authentication**

2235 SRP is a way of using a username and password to accomplish mutual authentication. The following
2236 ciphersuites defined in [draft-ietf-tls-srp-08] are supported.

2237 MUST:

2238 TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA

2239 SHOULD:

2240 TLS_SRP_SHA_WITH_AES_128_CBC_SHA

2241 TLS_SRP_SHA_WITH_AES_256_CBC_SHA

2242 **6.3.4 TLS SRP and X.509 Server Authentication**

2243 SRP can be combined with X.509 server authentication. The following ciphersuites defined in [draft-ietf-
2244 tls-srp-08] are supported.

2245 MUST:

2246 TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA

2247 SHOULD:

2248 TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA

2249 TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA

2250 7 DSS-Defined Identifiers

2251 The following sections define various URI-based identifiers. Where possible an existing URN is used to
2252 specify a protocol. In the case of IETF protocols the URN of the most current RFC that specifies the
2253 protocol is used (see [RFC 2648]). URI references created specifically for DSS have the following stem:

2254 urn:oasis:names:tc:dss:1.0:

2255 7.1 Signature Type Identifiers

2256 The following identifiers MAY be used as the content of the <SignatureType> optional input (see
2257 section 3.5.1).

2258 7.1.1 XML Signature

- 2259 • **URI:** urn:ietf:rfc:3275
- 2260 • This refers to an XML signature per [XMLDSIG].

2261 7.1.2 XML TimeStampToken

- 2262 • **URI:** urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken
- 2263 • This refers to an XML timestamp containing an XML signature, per section 5.1.

2264 7.1.3 RFC 3161 TimeStampToken

- 2265 • **URI:** urn:ietf:rfc:3161
- 2266 • This refers to an XML timestamp containing an ASN.1 TimeStampToken, per [RFC 3161].

2267 7.1.4 CMS Signature

- 2268 • **URI:** urn:ietf:rfc:3369
- 2269 • This refers to a CMS signature per [RFC 3852] or prior versions of CMS.

2270 7.1.5 PGP Signature

- 2271 • **URI:** urn:ietf:rfc:2440
- 2272 • This refers to a PGP signature per [RFC 2440].

2273

A. Use of Exclusive Canonicalization

2274 Exclusive Canonicalization of dereferenced and transformed data can be achieved by appending
2275 exclusive canonicalization as the last transform in the `<ds:Transforms>` element of
2276 `<TransformedData>` or `<DocumentHash>`.

2277 In the case of `<Document>` being used this can be done by adding exclusive canonicalization as the last
2278 transform in the `<ds:Transforms>` of a `<SignedReference>` pointing to that `<Document>`.

2279 By doing this the resulting data produced by the chain of transforms will always be octet stream data
2280 which will be hashed without further processing on a `<ds:Reference>` level by the server as indicated
2281 by basic processing section 3.3.1 step 1 b. and c.

2282 Another possibility to apply exclusive canonicalization on `<ds:Reference>` level is the freedom given to
2283 servers to apply additional transforms to increase robustness. This however implies that only trustworthy
2284 transformations are appended by a server.

2285 As in section 3.3.1 step 1 b an implementation can choose to use exclusive canonicalization: "...
2286 Transforms are applied as a server implementation MAY choose to increase robustness of the Signatures
2287 created. These Transforms may reflect idiosyncrasies of different parsers or solve encoding issues or the
2288 like. ..."

2289 In such a case that the exclusive canonicalization is to be included in the `<ds:Transforms>` as well (cf.
2290 section 3.3.1 step 1.d.v.)

2291 The standards default is however in line with [XMLDSIG] as indicated in the Note in section 3.3.1 step 1
2292 b.

2293 However after the server formed a `<ds:SignedInfo>` (section 3.3.1 step 3.) this information to be
2294 signed also needs to be canonicalized and digested, here [XMLDSIG] offers the necessary element
2295 `<ds:CanonicalizationMethod>` directly and can be used to specify exclusive canonicalization.

2296

B. More Complex <Response> Example

2297 To further explain the use of the <Response> element which is useful in cases where the DSS server is
2298 not able to respond with a special response type a more complex example is given in the following
2299 paragraph.

2300 Consider for example a client sends a <SignRequest> to a service that only supports
2301 <VerifyRequest>'s over plain HTTP (as opposed to protocols where some information could be
2302 derived from the header). As the service does not support <SignRequest>'s it has to either generate a
2303 <VerifyResponse> with a "bad message" result or fail at the HTTP layer. In the former case, the client
2304 will receive a response that does not correspond semantically to the request - it got a
2305 <VerifyResponse> to a <SignRequest>. This leaves both parties thinking that the other one is at
2306 fault.

2307

C. Acknowledgements

2308 The following individuals have participated in the creation of this specification and are gratefully
2309 acknowledged:

2310 **Participants:**

2311 Dimitri Andivahis, Surety
2312 Glenn Benson, JPMorganChase
2313 Juan Carlos Cruellas, *individual*
2314 Carlos Gonzalez-Cadenas, Netfocus, S.L
2315 Frederick Hirsch, Nokia
2316 Pieter Kasselmann, Cybertrust
2317 Andreas Kuehne, *individual*
2318 Konrad Lanz, Austria Federal Chancellery <Konrad.Lanz@iaik.tugraz.at>
2319 Tommy Lindberg, *individual*
2320 Paul Madsen, Entrust
2321 John Messing, American Bar Association
2322 Tim Moses, Entrust
2323 Trevor Perrin, *individual*
2324 Nick Pope, *Thales eSecurity*
2325 Rich Salz, DataPower
2326 Ed Shallow, Universal Postal Union