

2 **eXtended Signature Services (XSS) Profile**  
3 **of the OASIS Digital Signature Service**  
4 **(DSS)**

5 **Working Draft 07, 15 March 2006**

6 **Document identifier:**

7  
8 **Location:**

9 <http://www.oasis-open.org/committees/dss>

10 **Editor:**

11 Carlos González-Cadenas, netfocus

12 **Contributors:**

13 Marta Cruellas, CATCert

14 Francesc Oliveras, CATCert

15 Ignacio Alamillo, CATCert

16  
17 **Abstract:**

18 This profile extends the DSS protocol and its XAdES profiles to support several advanced operations  
19 regarding signature creation and verification.

20  
21 Additionally, this profile provides further detail on some DSS/XAdES aspects that can be useful

22 **Status:**

23 This is a **Working Draft** produced by the OASIS Digital Signature Service Technical Committee.  
24 Committee members should send comments on this draft to [dss@lists.oasis-open.org](mailto:dss@lists.oasis-open.org).

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

## Table of Contents

30	1	Introduction .....	4
31	1.1	Notation .....	4
32	1.2	Schema Organization and Namespaces.....	4
33	2	Profile Features.....	5
34	2.1	Identifier.....	5
35	2.2	Scope .....	5
36	2.2.1	Additions to the Signing Protocol.....	5
37	2.2.2	Additions to the Verifying Protocol.....	5
38	2.2.3	Common Additions.....	5
39	2.3	Relationship to Other Profiles .....	6
40	2.4	Signature Object.....	6
41	2.5	Transport Binding .....	6
42	2.6	Security Binding .....	6
43	3	Common Protocol Structures.....	8
44	3.1	Optional Inputs and Outputs .....	8
45	3.1.1	Optional Input <dss:ClaimedIdentity> .....	8
46	3.1.2	Optional Input <ReturnSignedResponse> / Optional Output <ResponseSignature> ....	9
47	3.1.3	Optional Input <Archive> / Optional Output <ArchiveInfo>.....	10
48	3.1.4	Optional Input <ReturnSignatureInfo> / Optional Output <SignatureInfo> .....	11
49	3.1.5	Optional Input <ReturnX509CertificateInfo> / Optional Output <X509CertificateInfo> ..	12
50	3.2	Result Codes.....	12
51	4	Profile of Signing Protocol.....	14
52	4.1	Optional Inputs and Outputs .....	14
53	4.1.1	Optional Input <dss:SignatureType> .....	14
54	4.1.2	Optional Input <xadp:SignatureForm>.....	15
55	4.1.3	Optional Input <dss:KeySelector> .....	15
56	4.1.4	Optional Input <dss:Properties> .....	15
57	4.1.5	Optional Input <CounterSignature> / Optional Output <dss:UpdatedSignature> .....	17
58	4.1.6	Optional Input <ParallelSignature> .....	18
59	4.2	Result Codes.....	19
60	5	Profile of Verifying Protocol.....	21
61	5.1	Optional Inputs and Outputs .....	21
62	5.1.1	Optional Input and Output <dss:VerificationTime>.....	21
63	5.1.2	Optional Input <SignaturePolicy> / Optional output <SignaturePolicyInfo> .....	21
64	5.1.3	Optional Input <Scheme> / Optional output <SchemeInfo> .....	24

65	5.1.4 Optional Input <X509CertificateValidationOptions> .....	25	
66	5.1.5 Optional Input <dss:ReturnUpdatedSignature> / Optional Output <dss:UpdatedSignature> .....	25	25
67	5.1.6 Optional Input <RequireQualifiedCertificate> .....	25	
68	5.2 Result Codes .....	26	
69	6 Identifiers.....	28	
70	6.1 Signature Properties Identifiers.....	28	
71	6.1.1 Signed Properties .....	28	
72	6.1.2 Unsigned Properties .....	29	
73	6.2 Signature Form Identifiers.....	29	
74	7 References.....	31	
75	7.1 Normative .....	31	
76	Appendix A. Guidelines for optional inputs that customize the addition of signature properties .....	32	
77	Appendix B. Management of Signed Responses as Electronic Records / Evidences.....	33	
78	Appendix C. Message Authentication using X509 Certificates .....	34	
79	Appendix D. Client Authentication using SAML Assertions.....	35	
80	Appendix E. Client Authentication using different password-based schemes .....	36	
81	Appendix F. Usage of Signature Policies in Signature Creation and Verification .....	37	
82	Appendix G. Extraction of attributes from signatures, certificates and other elements.....	38	
83	Appendix H. Revision History .....	41	
84	Appendix I. Notices.....	43	
85			

---

## 86 1 Introduction

### 87 1.1 Notation

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

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

95 `Listings of XSS schemas appear like this.`

### 96 1.2 Schema Organization and Namespaces

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

100 This schema is associated with the following XML namespace:

101 `urn:oasis:names:tc:dss:1.0:profiles:XSS`

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

103 Conventional XML namespace prefixes are used in the schema:

- 104 • The prefix `xss:` stands for the DSS core namespace [Core-XSD].
- 105 • The prefix `ds:` stands for the W3C XML Signature namespace [XMLSig].
- 106 • The prefix `xs:` stands for the W3C XML Schema namespace [Schema1].
- 107 • The prefix `saml11:` stands for the OASIS SAML 1.1 Schema namespace [SAMLCore1.1].
- 108 • The prefix `saml20:` stands for the OASIS SAML 2.0 Schema namespace [SAMLCore2.0].
- 109 • The prefix `wsse:` stands for OASIS Web Services Security [WSS].
- 110 • The prefix `xades:` stands for ETSI XML Advanced Electronic Signatures (XAdES) [XAdES].
- 111 • The prefix `xadp:` stands for XAdES Profiles of the OASIS Digital Signature Service [XAdES-DSS].
- 112 • The prefix `xsp:` stands for XML Format for Signature Policies [XMLSigPol].
- 113 • The prefix `tsl:` stands for Provision of Harmonized Trust Service Provider Status Information [TS 102 231].
- 114 • The prefix `archp:` stands for Signature Archive Profile of the OASIS Digital Signature Service [Archive-  
115 **DSS**].
- 116 • The prefix `xss:` or no prefix defaults to the namespace of the present document.

---

## 117 2 Profile Features

### 118 2.1 Identifier

119 urn:oasis:names:tc:dss:1.0:profiles:XSS

### 120 2.2 Scope

121 This document profiles the DSS XAdES profiles included in [XAdES-DSS-XML] and [CAAdES-DSS-XML].

#### 122 2.2.1 Additions to the Signing Protocol

- 123 • Creation of advanced electronic signatures based on a Signature Policy, as defined in [TR 102 038]  
124 or [TR 102 272]
- 125 • Archival of advanced electronic signatures after their creation, supporting the usage of Archival  
126 Policies.
- 127 • Creation of counter-signatures and parallel signatures.

#### 128 2.2.2 Additions to the Verifying Protocol

- 129 • X.509 Certificate Verification, allowing the clients to submit not only advanced electronic signatures or  
130 timestamps, but also X.509 Public-Key Certificates (PKCs) and X.509 Attribute Certificates (ACs),  
131 additionally allowing to customize how the server performs the certificate verification (algorithms and  
132 parameters).
- 133 • Support for Trust Service Provider status information lookup, by means of Trust Service Provider  
134 Status Lists (TSLs), as defined in [TS 102 231], in order to effectively enable scheme-based / cross-  
135 border transactions.
- 136 • Verification of advanced electronic signatures based on a Signature Policy, as defined in [TR 102  
137 038] or [TR 102 272].
- 138 • Archival of advanced electronic signatures after verification, in a similar way as described above for  
139 the Signing Protocol.
- 140 • Extraction of attributes contained in the signature objects (i.e. signatures, end entity certificate, ...),  
141 like the signer identity, the signing time or other useful information, specially useful when the clients of  
142 the signature server are also applications (i.e. performing authorization tests based on the attributes  
143 obtained from the response).
- 144 • Verification of qualified certificates.

#### 145 2.2.3 Common Additions

- 146 • Support for digitally signed responses that can be retained as evidences by the clients.
- 147 • Several authentication mechanisms are discussed in detail.

148 This profile is concrete, can be directly implemented, and MAY be further profiled.

## 149 2.3 Relationship to Other Profiles

150 This profile includes the features covered in the profiles included in the following table

Profile	Type	Description
<i>XML Advanced Electronic Signatures [XAdES-DSS-XML]</i>	CONCRETE	Support for the creation of [XAdES] signatures.
<i>CMS Advanced Electronic Signatures [CAAdES-DSS-XML]</i>	CONCRETE	Support for the creation of [CAAdES] signatures.
<i>XML Timestamping Profile of the OASIS Digital Signature Services [TST-DSS]</i>	CONCRETE	Support for the creation of CMS and XML timestamps.
<i>Signature Archive Profile of the OASIS Digital Signature Services [Archive-DSS]</i>	CONCRETE	Support for the archive of signatures.

## 151 2.4 Signature Object

152 In addition to the child elements defined in [DSS Core], the element `<dss:SignatureObject>` MAY  
153 contain one `<ds:X509Data>`, included in the `<dss:Other>` element.

154 When present, this element MUST include one or more `<ds:X509Certificate>` elements, containing one  
155 or more X.509 Public-Key Certificates (PKCs) and X.509 Attribute Certificates (ACs) conforming to  
156 [RFC3280] and [RFC3281], respectively, with the following restrictions

- 157 • exactly ONE end-entity X509 Public-Key Certificate can be included
- 158 • the included Attribute Certificates (if any) MUST be linked to the end-entity X509 Public-Key  
159 Certificate, following guidelines described in [RFC3281].

160 This element MUST only be included in a verify request message, and allows the client to request the  
161 verification of X.509 Certificates to the server.

## 162 2.5 Transport Binding

163 This profile does not constrain any transport binding defined in [DSSCore].

## 164 2.6 Security Binding

165 This profile does not constrain any security binding defined in [DSSCore].

166 A security analysis SHOULD be carried to assure that a proper combination of transport and security bindings  
167 is used according to the applicable security policy.

---

## 168 3 Common Protocol Structures

### 169 3.1 Optional Inputs and Outputs

170 None of the optional inputs specified in the [DSS Core] and [XAdES-DSS] are precluded in this abstract  
171 profile. It only constrains some of them and specifies additional optional inputs.

#### 172 3.1.1 Optional Input <dss:ClaimedIdentity>

173 The optional input <dss:ClaimedIdentity> MAY be present when the requested operation (i.e. signature  
174 production) requires the client to be authenticated, and the chosen underlying security binding does not fully  
175 authenticate the client

176 There are several cases for that behaviour

177 • when the requester is not directly the client, but is acting on behalf of the latter (i.e. the requester is a  
178 presentation component of a Distributed Signature-Creation Application (SCA), as defined in [CWA  
179 14170]).

180 In this case, the Signature Creation Application (SCA) MAY choose to use a security binding that  
181 authenticates itself to the server (which is desirable in order to restrict the SCAs that can request  
182 signatures on behalf of the signers), but, in this case, there is no signer information that can be  
183 considered as signer authentication data.

184 • when the signer credentials used with the underlying security binding (if any) are **not suitable** to fully  
185 authenticate the signer (i.e. the applicable security policy requires to authenticate the signer using  
186 some non-standard / proprietary authentication method not covered as a standard security binding).

187 • otherwise, when the signer credentials used with the underlying security binding are **not enough** to  
188 fully authenticate the signer (i.e. when the applicable security policy requires more than one  
189 authentication factor to consider the request as valid (i.e. TLS Client Authentication plus PIN or One-  
190 Time Password).

191 It is STRONGLY recommended to perform a security analysis of the authentication methods to prevent  
192 guessing, impersonation and replay attacks. Man-in-the-middle attacks are mitigated by using one of the  
193 security bindings detailed below in this profile.

194 Different client/message authentication schemes are described in annexes C, D and E.

195 **3.1.2 Optional Input <ReturnSignedResponse> / Optional Output**  
196 **<ResponseSignature>**

197 The <ReturnSignedResponse> element instructs the server to produce a response signed with its own  
198 key. Normally, this signed response that can be retained / archived by the client (signer) of the service as an  
199 evidence of the validation process.

200 The management of the response after its production falls under responsibility of the client requesting the  
201 signature. See Appendix B for some guidelines in the management of the signed responses.

202 Optionally, the client can request to the server to create the signature under one or more commitments, using  
203 <RequiredCommitments>

204 <RequiredCommitments> [Optional]

205 The commitments requested by the client to be taken by the server when issuing the signed response.  
206 Commitments used MAY include the ones defined in [XAdES] or [CAAdES], or any other specific /  
207 proprietary ones.

208  
209 When no required commitments are specified, it's STRONGLY recommended for the signed responses to  
210 be produced under, at least, a commitment that recognizes the creation the signature as requested by the  
211 client (normally referred as **Proof Of Origin** commitment, as specified in [XAdES] and [CAAdES]).

212  
213 If the requested commitments cannot be applied by the server when generating the signature, the server  
214 MUST reject the request using the minor code UnavailableCommitment.

215

216 The server MAY decide, attending to its configuration, to generate a signed response even when the client  
217 (signer) hasn't requested the generation.

218 Validation of signed responses (standard enveloped signatures) can be carried out directly with capabilities  
219 provided by the DSS Core Protocol (DSS Verifying Protocol), without any specific extensions.

```
220 <xs:element name="ReturnSignedResponse">  
221 <xs:complexType>  
222 <xs:sequence>  
223 <xs:element name="RequiredCommitments" minOccurs="0">  
224 <xs:complexType>  
225 <xs:sequence>  
226 <xs:element name="CommitmentType" type="xsp:CommitmentType"  
227 maxOccurs="unbounded"/>  
228 </xs:sequence>  
229 </xs:complexType>  
230 </xs:element>  
231 </xs:sequence>  
232 </xs:complexType>  
233 </xs:element>
```

234

235 The signature MUST be an enveloped [XAdES] signature included in the <ResponseSignature>  
236 covering, at least,

- 237
- the whole document where the signature is enveloped into (using an enveloped signature transform and an appropriate reference uri, like URI="").
  - its own `<xades:SignedProperties>` element, as described in [XAdES].
- 238
- 239
- 240

```
241 <xs:element name="ResponseSignature">
242   <xs:complexType>
243     <xs:sequence>
244       <xs:element ref="ds:Signature"/>
245     </xs:sequence>
246   </xs:complexType>
247 </xs:element>
```

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

### 249 3.1.3 Optional Input <Archive> / Optional Output <ArchiveInfo>

250 The <Archive> element MAY be used by the client to request the archival of the signature after its  
251 processing by the server. This option will normally be used by these clients that don't have the means to  
252 manage the produced signature by themselves or those that prefer relying on a trusted third-party to perform  
253 this signature management over time.

254 The <Archive> element MAY include the different options defined in the profile [Archive-DSS]

```
255 <xs:element name="Archive">
256   <xs:complexType>
257     <xs:sequence>
258       <xs:choice>
259         <xs:element ref="archp:ArchivePolicy" minOccurs="0"/>
260         <xs:element ref="archp:RetentionPeriod" minOccurs="0"/>
261       </xs:choice>
262       <xs:element ref="archp:UpdateSignature" minOccurs="0"/>
263       <xs:element ref="archp:ArchiveMode" minOccurs="0"/>
264     </xs:sequence>
265   </xs:complexType>
266 </xs:element>
```

267 The <ArchiveInfo> response MUST include the identifier associated to the archived object

```
268 <xs:element name="ArchiveInfo">
269   <xs:complexType>
270     <xs:sequence>
271       <xs:element name="ArchiveIdentifier"/>
272     </xs:sequence>
273   </xs:complexType>
274 </xs:element>
```

275 The result codes for the archive operations can be found in [Archive-DSS].

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

277 **3.1.4 Optional Input <ReturnSignatureInfo> / Optional Output <SignatureInfo>**

278 The <ReturnSignatureInfo> element MAY be used by the client to request the extraction of attributes  
279 from the signature being produced that may be useful for the client requesting the signature.

280 <AttributeDesignator> [One or More]

281 A designator that points to the attribute being requested.

```
282 <xs:element name="ReturnSignatureInfo">  
283 <xs:complexType>  
284 <xs:sequence>  
285 <xs:element name="AttributeDesignator" type="saml20:AttributeType" maxOccurs="unbounded"/>  
286 </xs:sequence>  
287 </xs:complexType>  
288 </xs:element>  
289
```

290 The <SignatureInfo> element is used to carry the requested attributes.

291 <Attribute> [One or More]

292 The requested attribute.

```
293 <xs:element name="SignatureInfo">  
294 <xs:complexType>  
295 <xs:sequence>  
296 <xs:element name="Attribute" type="saml20:AttributeType" maxOccurs="unbounded"/>  
297 </xs:sequence>  
298 </xs:complexType>  
299 </xs:element>  
300
```

301 Compliant servers MUST process requests in the following manner:

- 302 • when the signature attribute is not known by the server, the server MUST reject the request using the  
303 minor code `InvalidSignatureAttribute`.
- 304 • when the signature attribute is known by the server, but is not supported, the server MUST reject the  
305 request using the minor code `UnsupportedSignatureAttribute`.
- 306 • when the signature attribute is not included in the signature, the server MUST not include an empty  
307 property in the response.
- 308 • when there are no available attributes to return, the server MUST not return the <SignatureInfo>  
309 element.

310 See Appendix G for details about the usage and some predefined attributes for this optional input.

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

312 **3.1.5 Optional Input <ReturnX509CertificateInfo> / Optional Output**  
313 **<X509CertificateInfo>**

314 The <ReturnX509CertificateInfo> element MAY be used by the client to request the parsing and  
315 further extraction of attributes from the signer's end-entity certificate (if any) in signatures and timestamps, or  
316 the end-entity public-key certificate (when validating certificates), according to [RFC3280].

317 <AttributeDesignator> [One or More]

318 A designator that points to the attribute being requested.

```
319 <xs:element name="ReturnX509CertificateInfo">  
320 <xs:complexType>  
321 <xs:sequence>  
322 <xs:element name="AttributeDesignator" type="saml20:AttributeType" maxOccurs="unbounded"/>  
323 </xs:sequence>  
324 </xs:complexType>  
325 </xs:element>  
326
```

327 The <X509CertificateInfo> element is used to carry the requested attributes.

328 <Attribute> [One or More]

329 The requested attribute.

```
330 <xs:element name="X509CertificateInfo">  
331 <xs:complexType>  
332 <xs:sequence>  
333 <xs:element name="Attribute" type="saml20:AttributeType" maxOccurs="unbounded"/>  
334 </xs:sequence>  
335 </xs:complexType>  
336 </xs:element>  
337
```

338 Compliant servers MUST process requests in the following manner:

- 339
- 340 • when the certificate attribute is not known by the server, the server MUST reject the request using the minor code `InvalidCertificateAttribute`.
  - 341 • when the certificate attribute is known by the server, but is not supported, the server MUST reject the request using the minor code `UnsupportedCertificateAttribute`.
  - 342 • when the certificate attribute is not included in the certificate, the server MUST not include an empty property in the response.
  - 343 • when there are no available attributes to return, the server MUST not return the
  - 344 <X509CertificateInfo> element.
- 345

347 See Appendix G for details about the usage and some predefined attributes for this optional input.

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

349 **3.2 Result Codes**

350 Here are some result codes shared by the two protocol profiles.

351 The URN used for the <dss:ResultMajor> elements is described in **[DSSCore]**. The URN used for the  
 352 <dss:ResultMinor> elements **MUST** be  
 353 urn:oasis:names:tc:dss:1.0:profiles:XSS:resultminor: followed by the codes described  
 354 below.

<dss:ResultMajor>	<dss:ResultMinor>	Description
RequesterError	SignaturePolicyNotFound	The server is unable to find an appropriate signature policy using the identifier requested by the client (signer).
RequesterError	UnavailableCommitment	The server cannot issue a signed response under the requested commitment.
RequesterError	SignaturePropertiesNotSupported	The signature type does not support signature properties.
RequesterError	InvalidSignatureAttribute	The requested signature attribute is not known by the server.
ResponderError	UnsupportedSignatureAttribute	The requested signature attribute is known, but not supported by the server.
RequesterError	InvalidCertificateAttribute	The requested certificate attribute is not known by the server.
ResponderError	UnsupportedCertificateAttribute	The requested certificate attribute is known, but not supported by the server.
RequesterError	SignaturePoliciesNotSupported	The client has requested an operation over a non <b>[XAdES]</b> or <b>[CAAdES]</b> signature.

---

355 **4 Profile of Signing Protocol**

356 **4.1 Optional Inputs and Outputs**

357 **4.1.1 Optional Input <dss:SignatureType>**

358 This profile supports the following signature types

<b>Signatures</b> (BASE FORMAT)	<b>Identifier</b>	<b>Description</b>
(CMS)	urn:ietf:rfc:3852	CMS Signature, according to RFC 3852 [RFC3852].
(CMS)	http://uri.etsi.org/01733/v1.6.3#	CAdES Signature, according to ETSI TS 101 733 v1.6.3 [CADES].
(XMLDSIG)	urn:ietf:rfc:3275	XML Digital Signature, according to RFC 3275 [XMLSig].
(XMLDSIG)	http://uri.etsi.org/01903/v1.2.2#	XAdES Signature, according to ETSI TS 101 903 v1.2.2 [XAdES].
<hr/>		
<b>Timestamps</b> (BASE FORMAT)	<b>Identifier</b>	<b>Description</b>
(CMS)	urn:ietf:rfc:3161	CMS/CADES Timestamp, according to RFC 3161.
(XMLDSIG)	oasis:names:tc:dss:1.0:core: schema:XMLTimeStampToken	XML Timestamp, as defined in OASIS DSS Core.
(XMLDSIG)	oasis:names:tc:dss:1.0:core: schema:XAdESTimeStampToken	XAdES Timestamp, as defined in OASIS DSS Core, additionally protecting the signing certificate as described in XAdES.

359 If no <dss:SignatureType> is included in the request, the server MAY decide to create any type of  
360 signature based on its configuration.

#### 361 **4.1.2 Optional Input <xadp:SignatureForm>**

362 This optional input instructs the server to create a signature using one of the forms defined in both [CADES]  
363 and [XAdES]. Therefore, it can only be used when <dss:SignatureType> includes one of the following  
364 values

365 • <http://uri.etsi.org/01733/v1.6.3#>, for a [CADES] signature.

366 • <http://uri.etsi.org/01903/v1.2.2#>, for a [XAdES] signature.

367 For any other values, the server MUST reject the request using the minor code  
368 SignatureFormsNotSupported.

369 Valid signature forms for this profile are included in section 8.1.

#### 370 **4.1.3 Optional Input <dss:KeySelector>**

371 The server MUST authenticate the client (signer) previously to perform the lookup of the key to generate the  
372 signature (previously to the access to any key material). For that purpose, the server MUST use the  
373 authentication information obtained from the underlying security binding and/or the authentication information  
374 obtained from the optional input <dss:ClaimedIdentity>, as described above.

375 The server MAY additionally perform authorization checks (i.e. can the user access the selected private key?)  
376 for the referenced key, always within the key-space determined for the authenticated subject.

377 Additionally, the server MAY perform binding validity checks to assure that the binding between the signer  
378 (the entity identified by the attributes present in the <dss:ClaimedIdentity> element or in the underlying  
379 security binding) and the key is still valid.

380 When <dss:KeySelector> is present, it MUST contain a <ds:KeyInfo> including a valid pointer to the  
381 public key complementary to the client's signing private key. If the server cannot locate the key using this  
382 name, the server MUST reject the request using the minor code KeyNotFound\_InvalidIdentifier.

383 The optional input <dss:KeySelector> MUST appear when there are more than one applicable key for  
384 signature purposes associated to the client (signer).

385 When the <dss:KeySelector> element is not present, the server MUST obtain the only applicable key for  
386 signature purposes of the signer. When more than one key are applicable for signature purposes, the server  
387 MUST reject the request using the minor code KeyNotFound\_MoreThanOneKeyFound.

#### 388 **4.1.4 Optional Input <dss:Properties>**

389 The <dss:Properties> element instructs the server to add signed/unsigned signature properties to the  
390 signature. This profile further details the properties valid for inclusion and the information that MUST be sent  
391 to the server for processing, either

392 • the attribute identifier only, letting the server to add the value

393 • the attribute identifier and the value, only letting the server to place the attributes in their respective  
394 holders

395 The server MUST refuse to create the signature, using the minor code  
 396 SignaturePropertiesNotSupported in the following cases

- 397 • when the signature type requested is not [CAAdES] or [XAdES]
- 398 • when the selected attribute cannot be added to the [CAAdES] or [XAdES] signature (i.e.  
 399 IndividualObjectsTimestamp for a [CAAdES] signature)

400 **4.1.4.1 Signed Properties**

Identifier	Information Required
SigningTime	Attribute Identifier Only
SigningCertificate	Attribute Identifier Only
SignaturePolicyIdentifier	Attribute Identifier and Value
ContentIdentifier	Attribute Identifier and Value
ContentReference	Attribute Identifier and Value
DataObjectFormat	Attribute Identifier and Value
CommitmentTypeIndication	Attribute Identifier and Value(s)
SignatureProductionPlace	Attribute Identifier Only
SignerRole	Attribute Identifier and Value
AllDataObjectsTimestamp	Attribute Identifier Only
IndividualDataObjectsTimestamp	Attribute Identifier Only

401 All the identifiers MUST be prefixed by urn:oasis:names:tc:dss:1.0:profiles:XAdES:

402

403 **4.1.4.2 Unsigned Properties**

Identifier	Information Required
SignatureTimestamp	Attribute Identifier Only
CompleteCertificateRefs	Attribute Identifier Only
CompleteRevocationRefs	Attribute Identifier Only

AttributeCertificateRefs	Attribute Identifier Only
AttributeRevocationRefs	Attribute Identifier Only
SigAndRefsTimestamp	Attribute Identifier Only
RefsOnlyTimestamp	Attribute Identifier Only
CertificateValues	Attribute Identifier Only
RevocationValues	Attribute Identifier Only
ArchiveTimestamp	Attribute Identifier Only

404 All the identifiers MUST be prefixed by urn:oasis:names:tc:dss:1.0:profiles:XAdES:

405 When the signature policy or the commitment cannot be found, the server MUST refuse the request using  
406 minor codes SignaturePolicyNotFound and CommitmentNotFound, respectively.

#### 407 **4.1.5 Optional Input <CounterSignature> / Optional Output** 408 **<dss:UpdatedSignature>**

409 This element allows the client to request the creation of a countersignature over an existing signature. If the  
410 signature type requested is [CMS], [CAAdES] or [XAdES], the countersignature will be added to the  
411 countersignature unsigned attribute of the countersigned signature and returned in the  
412 <dss:UpdatedSignature> optional output.

413 The signature MUST be included in a <dss:Document> element inside the <dss:InputDocuments>  
414 element. The document containing the signature MUST be pointed using the attribute WhichDocument of the  
415 <CounterSignature> element.

416 Some restrictions apply to the countersignature creation

- 417 • Only countersignatures of the same type are allowed (i.e. no XML countersignatures over CMS  
418 signatures)
- 419 • When creating XML signatures, only <ds:Signature> elements can be passed as root of the  
420 <dss:Document> element.
- 421 • When creating CMS countersignatures, the <dss:Document> element within  
422 <dss:InputDocuments> MUST only contain a <dss:Base64Data> including the signature.

423

```
424 <xs:element name="CounterSignature">
425   <xs:complexType>
426     <xs:attribute name="WhichDocument" type="xs:IDREF" use="required"/>
427   </xs:complexType>
428 </xs:element>
```

#### 429 **4.1.5.1 Basic Processing forXML Signatures**

430 Three new steps 1.b0, 1.b1 and 1.b2 are inserted before 1.b in the section 3.3.1

- 431 1
- 432 b.0 The server parses the octet stream into NodeSetData (if not done before).
- 433 b.1 The server forms a `<ds:Reference>` pointing to the `<ds:SignatureValue>` element of the
- 434 `<ds:Signature>` included in the parsed document obtained from b.0.
- 435 b.1 The document containing the signature is removed from the set of unprocessed documents, so
- 436 it's not considered in the rest of the process.

437

438 A new step 4 is inserted at the end of the section 3.3.1

- 439 4 If the created signature is a **[XAdES]** signature, the created signature is inserted into the
- 440 CounterSignature unsigned attribute of the countersigned signature. The updated signature is returned to the
- 441 client using the `<dss:UpdatedSignature>` optional output.

#### 442 **4.1.5.2 Basic Processing for CMS Signatures**

443 The step 1 of the section 3.4 is replaced with the following

444 1

- 445 a The server decodes the signature included in the `<dss:Base64Data>` element and parses the
- 446 CMS Signature
- 447 b The server hashes the signature value of the countersigned signature

448

449 The step 2.b of the section 3.4 is modified as follows

450 2

- 451 b Add to the end of the paragraph "respecting the guidelines for countersignature creation as
- 452 described in section 11.4 of **[RFC3852]**."

453 The step 3 of the section 3.4 is replaced with the following

- 454 3 The server includes the created `SignerInfo` into the countersigned `SignedData`'s CounterSignature
- 455 unsigned attribute. The updated signature is returned to the client using the `<dss:UpdatedSignature>`
- 456 optional output.

#### 457 **4.1.6 Optional Input <ParallelSignature>**

458 This element allows the client to request the creation of a CMS `SignerInfo` over an existing `SignedData`

459 including its encapsulated content. The `SignerInfo` is included in the existing `SignedData` and returned

460 normally in the `<dss:SignatureObject>` of the `<dss:SignResponse>`. The encapsulated content

461 type MUST be `id-data`.

462

463 The `<dss:Document>` element within `<dss:InputDocuments>` MUST only contain a

464 `<dss:Base64Data>` including the signature in order to create the parallel signature.

465

466 

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

467 **4.1.6.1 Basic Processing**

468 The step 1 of the section 3.4 is replaced with the following

469 1

470 a The server decodes the signature included in the `<dss:Base64Data>` element and parses the  
471 CMS Signature

472 b The server hashes the encapsulated content included into the signature.

473

474 The step 3 of the section 3.4 is replaced with the following

475 3 The server includes the created `SignerInfo` into the decoded `SignedData` passed as input.

476 **4.2 Result Codes**

477

<code>&lt;dss:ResultMajor&gt;</code>	<code>&lt;dss:ResultMinor&gt;</code>	Description
RequesterError	KeyNotFound_InvalidIdentifier	The server cannot find a key using the key identifier passed in the request.
RequesterError	KeyNotFound_MoreThanOneKeyFound	The server has found more than one suitable key and cannot determine the key to use.
RequesterError	IncompatibleSignatureForms	The server has found that the signature form requested in the <code>&lt;xadp:SignatureForm&gt;</code> element and the one included in the signature policy referenced or included in the <code>&lt;SignaturePolicy&gt;</code> are incompatible.
RequesterError	SignatureFormsNotSupported	The client has selected a <code>&lt;dss:SignatureType&gt;</code> that does not support signature forms.
RequesterError	CommitmentNotFound	The selected commitment cannot be found by the server.



---

## 480 5 Profile of Verifying Protocol

### 481 5.1 Optional Inputs and Outputs

#### 482 5.1.1 Optional Input and Output <dss:VerificationTime>

483 The element <dss:VerificationTime> instructs the server to attempt to determine the signature's  
484 validity at the specified time, instead of the current time.

485 Depending on the kind of input to the <dss:VerifyRequest>, the behaviour of the server MAY vary, mainly  
486 determined by the kind of signatures to be validated (i.e. timestamps, certificates or CMS/XMLDSig  
487 signatures themselves).

488 The semantics for the different elements are

- 489 • for certificates, this verification time allows the client to request the verification of the status of the  
490 certificate when requested.
- 491 • for timestamps, this verification time allows the client to request the verification of the status of the  
492 timestamp when requested.
- 493 • for signatures, different cases arise
  - 494 ○ when the signing time is known and trusted, this parameter MAY have no effect and the  
495 signature SHOULD be verified using the trusted signing time.
  - 496 ○ when the signing time is not known, the server MAY perform the signature verification using  
497 this time

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

#### 499 5.1.2 Optional Input <SignaturePolicy> / Optional output <SignaturePolicyInfo>

500 The element <SignaturePolicy> MAY be present to request the verification of a signature against a  
501 specific signature policy.

502 There are several inputs to be taken into account when verifying signatures with policy information

- 503 • the signature policy included in the request (if any).
- 504 • the explicit policy included as a signed attribute at signature production (if any).
- 505 • the policy defined as the default policy in the server (if any), applicable when no policy is present in  
506 the request or in the signature.
- 507 • signature policies (if any) determined to be compatible with the policy included in the request or the  
508 signature by the service (signature policy mappings).

509 There are several combinatorial cases with the related inputs. A reference processing model is proposed  
510 below. Further profiles MAY describe other applicable processing models.

511

Request	Signature	Policy-Mapping	Default	Result
X	X	N/A	X	The service verifies the signature without checking against any policy.
X	X	N/A	V	The service verifies the signature against the default policy.
X	V	X	N/A	The service verifies the signature against the policy included in the signature.
X	V	V	N/A	The service verifies the signature against the policy included in the signature (or any of its policy mappings, if available).
V	X	N/A	N/A	The service verifies the signature against the policy included in the request.
V	X	V	N/A	The service verifies the signature against the policy included in the request (or any of its policy mappings, if available).
V	V	N/A	N/A	The service verifies the signature against the policy included in the request ( <b>overriding the one in the signature</b> ).
V	V	V	N/A	The service verifies the signature against the policy included in the request (or any of its policy mappings, if available) ( <b>overriding the one in the signature</b> ).

512 The element `<SignaturePolicyInfo>` MUST be returned when the server performs a signature  
513 verification using a policy, even when no `<SignaturePolicy>` is included in the request.

514 The element `<SignaturePolicy>` MUST contain the OID or URI uniquely identifying a signature policy  
515 installed in the server.

516 When the signature being validated is not **[CAAdES]** or **[XAdES]**, the server MUST reject the request using the  
517 minor code `SignaturePoliciesNotSupported`.

518 When the server cannot resolve the signature policy with the passed identifier MUST reject the request using  
519 the minor code `SignaturePolicyNotFound`.

520 If the client would like to enable policy mappings, the attribute `policyMappingsEnabled` SHALL be  
521 asserted accordingly.

522

```

523 <xs:element name="SignaturePolicy">
524   <xs:complexType>
525     <xs:complexContent>
526       <xs:extension base="xades:ObjectIdentifierType">

```

```

527     <xs:attribute name="allowPolicyMappings" type="xs:boolean" use="optional"
528     default="false"/>
529     </xs:extension>
530   </xs:complexContent>
531 </xs:complexType>
532 </xs:element>

```

533 After signature verification, the server MUST include a `<SignaturePolicyInfo>` element including details  
534 about the signature policy requested by the client (verifier).

535 `<SignaturePolicyIssuer>` [Required]

536 The issuer of the signature policy.

537 `<SignaturePolicyIdentifier>` [Required]

538 The unique identifier (URI or OID) of the signature policy.

539 `<SignaturePolicyDigestAlgorithm>` [Required]

540 The unique identifier (URI or OID) of the algorithm used to digest the signature policy.

541 `<SignaturePolicyDigestValue>` [Required]

542 The value of the selected digest algorithm applied over the signature policy, after using (if applicable)  
543 the chain of transforms present in the `<ds:Transforms>` element.

544 `<ds:Transforms>` [Optional]

545 The transform chain applied to the signature policy before calculating the hash.

546

```

547 <xs:element name="SignaturePolicyInfo">
548   <xs:complexType>
549     <xs:sequence>
550       <xs:element name="SignaturePolicyIssuer" type="xs:string"/>
551       <xs:element name="SignaturePolicyIdentifier"
552       type="xades:ObjectIdentifierType"/>
553       <xs:element name="SignaturePolicyDigestAlgorithm"
554       type="xades:ObjectIdentifierType"/>
555       <xs:element name="SignaturePolicyDigestValue" type="ds:DigestValueType"/>
556       <xs:element ref="ds:Transforms" minOccurs="0"/>
557     </xs:sequence>
558   </xs:complexType>
559 </xs:element>

```

560 The signature policy MAY require the server to return an updated signature (by addition of one or more  
561 unsigned attributes). In this case, the server MUST use the `<dss:UpdatedSignature>` optional output to  
562 include the result of the signature updating.

563 Regarding the `Type` attribute of the element `<dss:UpdatedSignature>`, it will be filled as follows

- 564 • When the signer has produced the signature under one or more commitments, the `Type` attribute  
565 MUST be the identifier of the commitment, and the signature returned will be the one updated using  
566 the rules associated to that commitment.

567 • When the signer hasn't produced the signature under any specific commitment, the `Type` attribute  
568 MUST not appear, and the signature returned will be the one updated using the common rules  
569 (using the *null* commitment).

570 This optional input MUST not be used in combination with `<dss:ReturnUpdatedSignature>`.

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

### 572 **5.1.3 Optional Input `<Scheme>` / Optional output `<SchemeInfo>`**

573 The `<Scheme>` element MAY be used to perform the verification of all the trust service providers (TSPs)  
574 involved in the production of the signature being verified against a specific supervision scheme (to determine  
575 if these providers are "approved" ("trusted") under this supervision scheme, following the semantics and  
576 guidelines defined in **[TS 102 231]**.

577 Normally, this task is related with the verification of the trust service providers (whose identifiers, in form of  
578 public keys or public-key certificates, are included in the signature (i.e. CAs, TSAs, AAs...)) against a TSP  
579 Status List (TSL).

580 The elements used within `<Scheme>` MUST be interpreted as described in **[TS 102 231]**.

581 `<SchemeName>` [Required]

582 The name of the scheme. This attribute is used to locate the scheme from the several schemes  
583 defined in the server.

```
584 <xs:element name="Scheme">  
585   <xs:complexType>  
586     <xs:sequence>  
587       <xs:element name="SchemeName" type="tsl:InternationalNamesType"/>  
588     </xs:sequence>  
589   </xs:complexType>  
590 </xs:element>
```

591

592 When the scheme cannot be found, the server MUST refuse the request using the minor code  
593 `SchemeNotFound`.

594 When the request contains a `<Scheme>` element, the server MUST respond with a `<SchemeValidation>`  
595 element containing useful information about the scheme and the specific TSL object used in the validation  
596 process.

597 Additionally, the server MAY include in the response a `<SchemeInfo>` element, without having received a  
598 `<Scheme>` element, when some other element caused the server to perform the verification of the involved  
599 trust-service providers against the scheme information provided by means of a TSL.

600 `<SchemeName>` [Required]

601 The name of the scheme.

602 `<TSLSequenceNumber>` [Required]

603 The sequential number of the TSL object used to validate the signature.

604 `<TSLDigestAlgorithm>` [Required]

605 The algorithm used to digest the TSL object.

606 <TSLDigestValue> [Required]

607 The value of the digest over the TSL object using the algorithm included above.

608

```
609 <xs:element name="SchemeInfo">
610   <xs:complexType>
611     <xs:sequence>
612       <xs:element name="SchemeName" type="tsl:InternationalNamesType"/>
613       <xs:element name="TSLSequenceNumber" type="xs:integer"/>
614       <xs:element name="TSLDigestAlgorithm" type="xades:ObjectIdentifierType"/>
615       <xs:element name="TSLDigestValue" type="ds:DigestValueType"/>
616     </xs:sequence>
617   </xs:complexType>
618 </xs:element>
```

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

#### 620 **5.1.4 Optional Input <X509CertificateValidationOptions>**

621 The <UseX509CertificateValidationOptions> element can be used to instruct the server the  
622 initialization parameters to be used when validating end-entity X509 Certificates as defined in [RFC3280].  
623 This optional input is not valid when verifying signatures or timestamps.

624 <xs:element name="X509CertificateValidationOptions" type="xsp:CertificateTrustTreesType"/>

#### 625 **5.1.5 Optional Input <dss:ReturnUpdatedSignature> / Optional Output 626 <dss:UpdatedSignature>**

627 The server MUST refuse to update the signature, using the minor code  
628 SignaturePropertiesNotSupported when the signature type requested is not [CAAdES] or [XAdES].

629 The server MAY return a <dss:UpdatedSignature> even when the client hasn't requested the updating  
630 of a signature. For example, when validating a XAdES or CAAdES signature, the server MAY decide to return  
631 an (XAd)ES-T or (XAd)ES-C updated signature.

632

633 This optional input MUST not be used in combination <SignaturePolicy>.

634 Valid signature forms that can be used for updating are covered in section 6.2.

#### 635 **5.1.6 Optional Input <RequireQualifiedCertificate>**

636 The <RequireQualifiedCertificate> element can be used to instruct the server to check that the  
637 certificate used to create the signature (or the certificate itself when validating certificates) is a qualified  
638 certificate (according to the EC Directive on Electronic Signatures).

639 The server MUST check for a valid qualified certificate according to [RFC3739].

640 When used with <UseSchemeValidation>, the server MUST check that the end-entity certificate's CA is  
641 listed in the schema as an issuer for qualified certificates.

642

643 **5.2 Result Codes**

<dss:ResultMajor>	<dss:ResultMinor>	Description
RequesterError	SchemeNotFound	The referred scheme cannot be found by the server.

644  
645  
646  
647  
648  
649  
650  
The following result codes MAY be returned

651 by the server when validating certificates

652

<dss:ResultMajor>	<dss:ResultMinor>	Description
Success	valid:certificate:Definitive	The certificate is valid.
Success	valid:certificate:Temporal	The certificate is valid, but another validation MAY be needed in order to ensure proper propagation of the revocation information.
Success	invalid:certificate:OnHold	The certificate is not valid, as it's on hold.
Success	invalid:certificate:Revoked	The certificate is not valid, as it's revoked.
Success	invalid:certificate:Expired	The certificate is not valid, as it's expired.
Success	invalid:certificate:NotYetValid	The certificate is not yet valid, as its static validity period has not started.

Success	unknown:certificate:NoCertificatePathFound	The server can't find any valid certificate path.
Success	unknown:certificate:PathValidationFails	The certificate path is not valid.
Success	unknown:certificate:RevocationStatusInfoNotFound	Cannot obtain revocation status information.
Success	unknown:certificate:UntrustedRevocationStatusInfo	The obtained revocation status information cannot be trusted
Success	unknown:certificate:BadCertificateFormat	The certificate format is invalid.
Success	unknown:certificate:BadCertificateSignature	The certificate signature is invalid.

653

654

---

655 **6 Identifiers**

656 **6.1 Signature Properties Identifiers**

657 **6.1.1 Signed Properties**

658 The Signed Signature Properties supported in this profile (based on the properties defined in [CAAdES-DSS]  
659 and [XAdES-DSS]) are:

Identifier	[XAdES-DSS] Signature Property	[CAAdES-DSS] Signature Property
SigningTime	SigningTime	SigningTime
SigningCertificate	SigningCertificate	SigningCertificate OtherSigningCertificate
SignaturePolicyIdentifier	SignaturePolicyIdentifier	SignaturePolicyIdentifier
ContentIdentifier	N/A	ContentIdentifier
ContentReference	N/A	ContentReference
DataObjectFormat	DataObjectFormat	ContentHints
CommitmentTypeIndication	CommitmentTypeIndication	CommitmentTypeIndication
SignatureProductionPlace	SignatureProductionPlace	SignerLocation
SignerRole	SignerRole	SignerAttributes
AllDataObjectsTimestamp	AllDataObjectsTimestamp	ContentTimestamp
IndividualDataObjectsTimestamp	IndividualDataObjectsTimestamp	N/A

660

661 NOTES:

- 662 • The Identifiers MUST be prefixed by urn:oasis:names:tc:dss:1.0:profiles:XAdES:.
- 663 • The server MUST decide between SigningCertificate and OtherSigningCertificate
- 664 using the criteria defined in [CAAdES].

665

666 **6.1.2 Unsigned Properties**

Identifier	[XAdES] Signature Property	[CAAdES] Signature Property
CounterSignature	CounterSignature	CounterSignature
SignatureTimestamp	SignatureTimestamp	SignatureTimestamp
CompleteCertificateRefs	CompleteCertificateRefs	CompleteCertificateRefs
CompleteRevocationRefs	CompleteRevocationRefs	CompleteRevocationRefs
AttributeCertificateRefs	AttributeCertificateRefs	AttributeCertificateRefs
AttributeRevocationRefs	AttributeRevocationRefs	AttributeRevocationRefs
SigAndRefsTimestamp	SigAndRefsTimestamp	ESCTimestamp
RefsOnlyTimestamp	RefsOnlyTimestamp	TimestampedCertsCRLs
CertificateValues	CertificateValues	CertificateValues
RevocationValues	RevocationValues	RevocationValues
ArchiveTimestamp	ArchiveTimestamp	ArchiveTimestamp

667

668 NOTES:

- 669 • The Identifiers MUST be prefixed by urn:oasis:names:tc:dss:1.0:profiles:XAdES:.

670

671

672

673

674

675

676

677

678

679

680 **6.2 Signature Form Identifiers**

681

XAdES-BES BES	urn:oasis:names:tc:dss:1.0:profiles:XAdES:forms:BES
XAdES-EPES EPES	urn:oasis:names:tc:dss:1.0:profiles:XAdES:forms:EPES
XAdES-T ES-T	urn:oasis:names:tc:dss:1.0:profiles:XAdES:forms:ES-T
XAdES-C ES-C	urn:oasis:names:tc:dss:1.0:profiles:XAdES:forms:ES-C
XAdES-X Type 1 ES-X Type 1	urn:oasis:names:tc:dss:1.0:profiles:XAdES:forms:ES-X-1
XAdES-X Type 2 ES-X Type 2	urn:oasis:names:tc:dss:1.0:profiles:XAdES:forms:ES-X-2
XAdES-X-L Type 1 ES-X-L Type 1	urn:oasis:names:tc:dss:1.0:profiles:XAdES:forms:ES-X-L-1
XAdES-X-L Type 2 ES-X-L Type 2	urn:oasis:names:tc:dss:1.0:profiles:XAdES:forms:ES-X-L-2
XAdES-A ES-X-A	urn:oasis:names:tc:dss:1.0:profiles:XAdES:forms:ES-A

682

683

---

684 **7 References**

685 **7.1 Normative**

686 [TO BE DONE]

687

688

689

690

691

692

693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714

---

## Appendix A. Guidelines for optional inputs that customize the addition of signature properties

Some optional inputs, like `<xadp:SignatureForm>`, `<dss:Properties>` or `<dss:AddTimestamp>`, among other possible ones, customize the signed and unsigned properties/attributes included to the signature.

In the practice, it's possible to have several cases (bounded by space determined by the combinations of the former optional inputs) where there are several optional inputs (i.e. a form and a policy, a policy and some properties, ...). In these cases, different implementations MAY choose to implement different strategies, commonly

- try to accomplish requirements imposed by each one of the inputs, commonly by performing an union of the signature property sets expressed (i.e. signature policies and properties) or implied (i.e. forms) by the different inputs.
- process only one source of properties per request, and therefore refusing these requests including more than one of these optional inputs, by using `IncompatibleSignatureForms` minor code.

Additionally, there are some situations that prevent usage of properties

- in these signatures that doesn't support properties in the practice (i.e. **[RFC3275]** signatures define the `<ds:SignatureProperties>` but does not define any property).
- using signature properties defined for one signature type with another signature type (i.e. using signature properties defined in **[XAdES]** over a **[RFC3275]** signature).

In these cases the server MUST refuse the request, using a `SignaturePropertiesNotSupported` minor code.

---

715 **Appendix B. Management of Signed Responses as**  
716 **Electronic Records / Evidences**

717 As the signed responses are themselves electronic signatures, a key issue for the clients of the signature  
718 lifecycle management services is the retention period of the responses as electronic records / evidences,  
719 falling normally under two cases

720

721 • short-term signatures: the retention period is lower than the lifetime of the server's signing certificate  
722 (or the server's signing key). In this case, no further actions are needed to ensure non-repudiation of  
723 the signed response.

724

725 • long-term signatures: the retention period is greater than the lifetime of the server's signing certificate  
726 (or the server's signing key). In this case, further actions are needed to ensure non-repudiation of the  
727 signed response, like

728

729 ○ updating the signature to an adequate form that can survive threats like CA key compromise  
730 or end-of-life of cryptographic algorithms, using, for example, the verifying protocol defined in  
731 DSS and further profiled in this document.

732

733 ○ using non-repudiation or long-term archive services, using, for example, the archive protocol  
734 defined in **[Archive-DSS]**

735

736 That is, clients SHOULD evaluate the retention requirements imposed in their business or legal environments  
737 in order to mitigate the risk of a possible repudiation for the digital signatures applied over the responses.

---

738 **Appendix C. Message Authentication using X509**  
739 **Certificates**

740 Message authentication using X.509 Certificates as security tokens can be obtained by digitally signing the  
741 whole request message using the client private key as a proof of possession for the public key certified in the  
742 X.509 Certificate included into the signature.

743 The optional input `<dss:ClaimedIdentity>` MUST include the following

744 • the `<dss:Name>` element MUST include a X.509 Subject Name in the `Format` attribute following the  
745 conventions described in [SAMLCore1.1].

746 • the `<SupportingInfo>` child element MUST contain a `<ds:Signature>` element including at  
747 least one reference covering the whole document (`URI=""`) and an enveloped transform.

748 The `RequestID` attribute included in the request MUST be present in order to prevent replay attacks.  
749 Compliant servers MUST apply reasonable measures to prevent those attacks based on this identifier.

750 Processing rules in the server MUST include the following checks

751 • check the cryptographic validity of the signature and its coverage

752 • check that there is a trusted and valid binding between the public key included in the signature and  
753 the entity represented by the enclosed `<dss:Name>` (i.e. by checking the X.509 Certificate included  
754 in the signature or checking the validity of the binding against an XKMS [XKMS] server)

755 The details about the criteria and method of trust establishment in the X.509 Certificate (i.e. accepted  
756 certificate classes or types, revocation status, ...) or the XKMS binding are implementation specific,  
757 and therefore not covered in this profile.

758

---

## 759 Appendix D. Client Authentication using SAML 760 Assertions

761 Client Authentication using SAML Assertions as security tokens can be easily obtained by including a valid  
762 SAML Assertion into the DSS Request Message. Unfortunately, this approach has several weaknesses that  
763 can lead to well-known security attacks

764 • linking the SAML assertion to the request message (to obtain message authentication) is not  
765 straightforward and require additional mechanisms

766 • guaranteeing that the holder of the assertion is the same subject as the one included in the assertion  
767 is also very difficult

768 Usage of additional secure transport bindings, like TLS, is highly recommended.

769 The optional input `<dss:ClaimedIdentity>` MUST include the following

770 • the `<dss:Name>` element MUST include a X.509 Subject Name or an Email Address in the `Format`  
771 attribute following the conventions described in **[SAMLCore1.1]**.

772 • the `<SupportingInfo>` child element MUST contain a valid SAML 1.1 **[SAMLCore1.1]** or SAML  
773 2.0 **[SAMLCore2.0]** Assertion, carrying one Authentication Statement.

774 Processing rules in the server MUST include the following checks

775 • check the cryptographic validity of the assertion

776 • check that there is a trusted authentication statement where the subject of the assertion is the same  
777 as the one enclosed in the `<dss:Name>`.

778 The details about the criteria and method of trust establishment in the SAML Assertion (i.e. accepted  
779 assertion issuers, accepted authentication methods, accepted signature certificates used when  
780 digitally signing the assertion, assertion processing rules ...) are implementation specific, and  
781 therefore not covered in this profile.

782

---

783 **Appendix E. Client Authentication using different**  
784 **password-based schemes**

785 Client Authentication using password-based information as security tokens can be obtained by including  
786 password information into the DSS Request Message. The design criteria of the underlying password-based  
787 scheme is critical to prevent several known security attacks, like

- 788
- impersonation, by eavesdropping the message and obtaining the password information
- 789
- replay attacks, because of the limitations of the password schemes to uniquely link the password  
790 information to the message
- 791
- dictionary attacks, due to the limited combinations used by users when choosing their passwords

792 It's recommended to use password schemes that are designed to be resistant to these security attacks  
793 (among others). Usage of additional secure transport bindings, like TLS, is highly recommended.

794 The optional input `<dss:ClaimedIdentity>` MUST include the following

- 795
- the `<dss:Name>` element MUST include a X.509 Subject Name or an Email Address in the `Format`  
796 attribute following the conventions described in **[SAMLCore1.1]**.
- 797
- the `<SupportingInfo>` child element MUST contain a Security Token as defined in OASIS Web  
798 Services Security **[WSS]** like **[WSS-Username]**.

799 The WSS Username profile provides can accommodate virtually any kind of passwords or PIN Code  
800 schemes, like clear text, digested passwords, Secure Remote Password **[RFC2945]**, and other one  
801 time password schemes like S/KEY, as defined in **[RFC1760]** and One-Time Password System, as  
802 defined in **[RFC 2289]**.

803 Processing rules in the server are scheme dependent, and therefore not covered by this profile.

804

---

805 **Appendix F. Usage of Signature Policies in Signature**  
806 **Creation and Verification**

807 This profile allows the creation and verification of signatures using signature policies as defined in [ETSI TR  
808 102 238] or [ETSI TR 102 272], by means of signature policy identifiers related to signature policies  
809 previously installed in the server.

810 The creation and verification of signatures will only work when creating / verifying [XAdES] or [CAAdES]  
811 signatures.

812 The creation is managed using the <dss:Properties> optional input, by including the appropriate  
813 SignaturePolicyIdentifier attribute and its identifier value, and additionally by including one or more  
814 CommitmentTypeIndication when needed.

815 When dealing with verification, two cases arise

- 816 • for EPES signatures, the server MUST verify the signature using the indicated policy.
- 817 • for BES signatures, or when there's a need to override the signature policy, the verification can be  
818 performed by means of the <SignaturePolicy> optional input, as described in the section 5.1.2.

819 **Appendix G. Extraction of attributes from signatures,**  
 820 **certificates and other elements**

821 It's a common need for clients of DSS, especially in the verification services, to obtain different information  
 822 about the signature being verified or even the signing certificate included in the signature.

823 This information is normally used by applications calling these services, in order to show it to the end-users,  
 824 or to perform authentication / authorization operations.

825 This profile includes the optional inputs `<ReturnSignatureInfo>` and  
 826 `<ReturnX509CertificateInfo>`, that MAY be used by clients to request the extraction of different  
 827 information from the signatures or signing certificates.

828 These optional inputs and their correspondent outputs use the `<saml20:Attribute>` included in the  
 829 **[SAMLCore2.0]** specification. Its schema definition is reproduced below for convenience.

```

830
831 <xs:complexType name="AttributeType">
832   <xs:sequence>
833     <xs:element ref="saml20:AttributeValue" minOccurs="0" maxOccurs="unbounded"/>
834   </xs:sequence>
835   <xs:attribute name="Name" type="xs:string" use="required"/>
836   <xs:attribute name="NameFormat" type="xs:anyURI" use="optional"/>
837   <xs:attribute name="FriendlyName" type="xs:string" use="optional"/>
838   <xs:anyAttribute namespace="##other" processContents="lax"/>
839 </xs:complexType>
840
841 <element name="AttributeValue" type="xs:anyType" nillable="true"/>
  
```

842  
 843 The attribute requests MUST not contain any `<saml20:AttributeValue>` element, as they are only  
 844 requests for attributes. The responses MUST contain, apart from the attributes received in the request, one or  
 845 more values, following the guidelines described in **[SAMLCore2.0]**, section 2.7.3.1 (especially in those  
 846 aspects regarding typing of the elements included as attribute values).

847 This profile includes several useful attributes for extracting information from the signatures and certificates. As  
 848 the attribute names defined in this profile are URIs, the `saml20:NameFormat` attribute MUST contain the  
 849 value `urn:oasis:names:tc:SAML:2.0:attrname-format:uri`, as described in **[SAMLCore2.0]**,  
 850 section 8.2.2.

851  
 852 The attributes defined in this profile to be used with `<ReturnSignatureInfo>` are defined below. All the  
 853 attributes defined below MUST be prefixed with  
 854 `urn:oasis:names:tc:dss:1.0:profiles:XSS:signatureAttributes`  
 855

Attribute	Return Type	Description
DigestAlgorithm	xades:ObjectIdentifier	The algorithm (OID or URI) used to calculate the digest over the content being signed.
DigestEncryptionAlgorithm	xades:ObjectIdentifier	The encryption algorithm (OID or URI) used over the digest to produce the

		signature.
SignatureAlgorithm	xades:ObjectIdentifier	The signature algorithm(OID or URI) (digest algorithm plus encryption algorithm) used to produce the signature.
SignatureValue	xs:base64Binary	The result of applying the signature algorithm over the content being signed.

856

857 Additionally it's possible to request the extraction of the signature properties defined in section 6.1, using the  
858 identifier defined there. The types of the returned elements MUST be

859

- 860 • For XAdES signatures, their correspondent schema types, as defined in [XAdES-XSD].
- 861 • For CAdES signatures, xs:base64Binary.

862

863 The attributes defined in this profile to be used with <ReturnX509CertificateInfo> are defined below.

864 All the attributes defined below MUST be prefixed with  
865 urn:oasis:names:tc:dss:1.0:profiles:XSS:certificateAttributes.

866

Attribute	Return Type	Description
Version	xs:integer	The version of the certificate.
SerialNumber	xs:integer	The serial number of the certificate.
Signature	xs:base64Binary	The X509 certificate's signature.
SignatureAlgorithm	xs:string	The algorithm used to sign the certificate
IssuerDistinguishedName	xs:string	The issuer distinguished name, formatted as described in [MLSig] .
IssuerDistinguishedName:XXX	xs:string	The <i>AttributeValue</i> of the selected <i>AttributeType</i> extracted from the issuer distinguished name, from the ones described in [X.500] (i.e. commonName).  In case of multiple appearances, they are returned in the given order, separated by commas.
SubjectDistinguishedName	xs:string	The subject distinguished name, formatted as described in [XMLSig] .
SubjectDistinguishedName:XXX	xs:string	The <i>AttributeValue</i> of the selected <i>AttributeType</i> extracted from the subject distinguished name, from the ones described in [X.500] (i.e. commonName).  In case of multiple appearances, they are returned in the given order, separated by

		commas.
NotBefore	xs:dateTime	The validity start date for the certificate.
NotAfter	xs:dateTime	The validity end date for the certificate.
SubjectPublicKeyAlgorithm	xs:string	The algorithm the key was generated with.
SubjectPublicKey	xs:base64Binary	The certificate's public key.
KeyUsages	xs:string	The list of key usages (separated by commas) of the certificate (i.e. digitalSignature)
IssuerEmail	xs:string	The issuer's email address, first looking for the one in the issuer alternative names, and after looking in the E attribute in the subject (for backwards-compatibility).
SubjectEmail	xs:string	The subject's email address, first looking for the one in the issuer alternative names, and after looking in the E attribute in the subject (for backwards-compatibility).
CertificatePolicies	xs:string	The list of certificate policies (separated by commas) of the policies the certificate is issued under
Extension:XXX	xs:base64Binary	The ASN.1 value, DER-Encoded of the extension XXX.  Valid extension names can be found in <b>[RFC3280]</b> .

867  
868  
869  
870

## Appendix H. Revision History

Rev	Date	By Whom	What
wd01	26/12/2005	Carlos González-Cadenas	Initial Version
wd02	11/01/2006	Carlos González-Cadenas	Changes in the way of handling attribute extraction from signatures and certificates. Change affects to the <ReturnSignatureInfo> and <ReturnX509CertificateInfo> elements. Addition of Annex G.
wd03	23/01/2006	Carlos González-Cadenas	Minor corrections in the SAML Authentication appendix. Minor corrections in the <ReturnX509CertificateInfo> section. Fixed namespace typos for namespaces with prefixes dss and xadp (some of them were not present)
wd04		Carlos González-Cadenas	Type change from xades:ObjectIdentifier to xs:string in Appendix G, Table 2.  Clarifications in the Appendix G about the attribute prefix usage.
wd05		Carlos González-Cadenas	Added result codes for certificate validation. Added some useful certificate attributes (like emails, ...)
wd06		Carlos González-Cadenas	Revised <SignaturePolicy> and <ReturnUpdatedSignature> optional inputs to include their mutual exclusion from a <dss:VerifyRequest> and to include some wording about the usage of the <dss:UpdatedSignature> to return the signatures updated due to the signature policy (including multiple commitment case).

Rev	Date	By Whom	What
wd07		Carlos González-Cadenas	Updated the <dss:UpdatedSignature> to include the cases when we're returning updated signatures as a result of validating a CAdES or XAdES signature and updating these to ES-T / ES-C, as described in the CAdES or XAdES specifications.
wd08		Carlos González-Cadenas	Simplified the <dss:VerificationTime> and aligned to the DSS, but introducing some detail about the processing.

---

872 **Appendix I. Notices**

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

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

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

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

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

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

899