eXtended Signature Services (XSS) Profile of the OASIS Digital Signature Service (DSS)

Working Draft 04, 24 January 2006

Document identifier:

Location:
http://www.oasis-open.org/committees/dss

Editor:
Carlos González-Cadenas, netfocus

Contributors:
Marta Cruellas, CATCert
Francesc Oliveras, CATCert
Ignacio Alamillo, CATCert

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

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

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

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

Table of Contents

1 Introduction

1.1 Notation

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

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

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

1.2 Schema Organization and Namespaces

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

This schema is associated with the following XML namespace:

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

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

Conventional XML namespace prefixes are used in the schema:

- The prefix `xss:` stands for the DSS core namespace **[Core-XSD]**.
- The prefix `ds:` stands for the W3C XML Signature namespace **[XMLSig]**.
- The prefix `xs:` stands for the W3C XML Schema namespace **[Schema1]**.
- The prefix `saml11:` stands for the OASIS SAML 1.1 Schema namespace **[SAMLCore1.1]**.
- The prefix `saml20:` stands for the OASIS SAML 2.0 Schema namespace **[SAMLCore2.0]**.
- The prefix `wsse`: stands for OASIS Web Services Security **[WSS]**.
- The prefix xades: stands for ETSI XML Advanced Electronic Signatures (XAdES) **[XAdES]**.
- The prefix xadp: stands for XAdES Profiles of the OASIS Digital Signature Service **[XAdES-DSS]**
- The prefix xsp: stands for XML Format for Signature Policies **[XMLSigPol]**.
- The prefix tsl: stands for Provision of Harmonized Trust Service Provider Status Information **[TS 102 231]**.
- The prefix archp: stands for Signature Archive Profile of the OASIS Digital Signature Service **[Archive-DSS]**.

122 • The prefix xss: or no prefix defaults to the namespace of the present document.

2 Profile Features

2.1 Identifier

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

2.2 Scope

This document profiles the DSS XAdES profiles included in **[XAdES-DSS-XML]** and **[CAdES-DSS-XML]**.

2.2.1 Additions to the Signing Protocol

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

2.2.2 Additions to the Verifying Protocol

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

2.2.3 Common Additions

- Support for digitally signed responses that can be retained as evidences by the clients.

- Several authentication mechanisms are discussed in detail.

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

2.3 Relationship to Other Profiles

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

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

2.4 Signature Object

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

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

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

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

2.5 Transport Binding

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

2.6 Security Binding

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

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

3 Common Protocol Structures

3.1 Optional Inputs and Outputs

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

3.1.1 Optional Input <dss:ClaimedIdentity>

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

There are several cases for that behaviour

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

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

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

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

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

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

3.1.2 Optional Input <ReturnSignedResponse> / Optional Output <ResponseSignature>

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

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

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

`<RequiredCommitments>` [Optional]

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

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

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

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

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

```
     <xs:element name="ReturnSignedResponse">
     <xs:complexType>
        <xs:sequence>
           <xs:element name="RequiredCommitments" minOccurs="0">
              <xs:complexType>
                 <xs:sequence>
                    <xs:element name="CommitmentType"
type="xsp:CommitmentType" maxOccurs="unbounded"/>
                 </xs:sequence>
              </xs:complexType>
           </xs:element>
        </xs:sequence>
     </xs:complexType>
     </xs:element>
```

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

- 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]**.

```
<xs:element name="ResponseSignature">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="ds:Signature"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

This optional input is allowed in multi-signature verification.

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

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

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

```
<xs:element name="Archive">
   <xs:complexType>
      <xs:sequence>
         <xs:choice>
            <xs:element ref="archp:ArchivePolicy" minOccurs="0"/>
            <xs:element ref="archp:RetentionPeriod" minOccurs="0"/>
         </xs:choice>
         <xs:element ref="archp:UpdateSignature" minOccurs="0"/>
         <xs:element ref="archp:ArchiveMode" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

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

```
<xs:element name="ArchiveInfo">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="ArchiveIdentifier"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

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

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

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

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

`<AttributeDesignator>` [One or More]

A designator that points to the attribute being requested.

```
<xs:element name="ReturnSignatureInfo">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="AttributeDesignator" type="saml20:AttributeType"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

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

`<Attribute>` [One or More]

The requested attribute.

```
<xs:element name="SignatureInfo">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Attribute" type="saml20:AttributeType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Compliant servers MUST process requests in the following manner:

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

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

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

3.1.5 Optional Input <ReturnX509CertificateInfo> / Optional Output <X509CertificateInfo>

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

`<AttributeDesignator>` [One or More]

A designator that points to the attribute being requested.

```
<xs:element name="ReturnX509CertificateInfo">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="AttributeDesignator" type="saml20:AttributeType"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

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

`<Attribute>` [One or More]

The requested attribute.

```
<xs:element name="X509CertificateInfoInfo">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Attribute" type="saml20:AttributeType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Compliant servers MUST process requests in the following manner:

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

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

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

3.2 Result Codes

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

The URN used for the `<dss:ResultMajor>` elements is described in **[DSSCore]**. The URN used for the `<dss:ResultMinor>` elements MUST be `urn:oasis:names:tc:dss:1.0:profiles:XSS:resultminor:` followed by the codes described 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 **[XAdES]** or **[CAdES]** signature.

4 Profile of Signing Protocol

4.1 Optional Inputs and Outputs

4.1.1 Optional Input <dss:SignatureType>

This profile supports the following signature types

Signatures (BASE FORMAT)	**Identifier**	**Description**
(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]**.
Timestamps (BASE FORMAT)	**Identifier**	**Description**
(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.

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

4.1.2 Optional Input <xadp:SignatureForm>

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

- http://uri.etsi.org/01733/v1.6.3#, for a **[CAdES]** signature.
- http://uri.etsi.org/01903/v1.2.2#, for a **[XAdES]** signature.

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

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

4.1.3 Optional Input <dss:KeySelector>

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

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

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

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

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

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

4.1.4 Optional Input <dss:Properties>

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

- the attribute identifier only, letting the server to add the value
- the attribute identifier and the value, only letting the server to place the attributes in their respective holders

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

- when the signature type requested is not **[CAdES]** or **[XAdES]**
- when the selected attribute cannot be added to the **[CAdES]** or **[XAdES]** signature (i.e. IndividualObjectsTimestamp for a **[CAdES]** signature)

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`

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

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

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

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

4.1.5 Optional Input <CounterSignature> / Optional Output <dss:UpdatedSignature>

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

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

Some restrictions apply to the countersignature creation

- Only countersignatures of the same type are allowed (i.e. no XML countersignatures over CMS signatures)
- When creating XML signatures, only `<ds:Signature>` elements can be passed as root of the `<dss:Document>` element.

- When creating CMS countersignatures, the <dss:Document> element within <dss:InputDocuments> MUST only contain a <dss:Base64Data> including the signature.

```
<xs:element name="CounterSignature">
   <xs:complexType>
      <xs:attribute name="WhichDocument" type="xs:IDREF" use="required"/>
   </xs:complexType>
</xs:element>
```

4.1.5.1 Basic Processing for XML Signatures

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

1

b.0 The server parses the octet stream into NodeSetData (if not done before).

b.1 The server forms a <ds:Reference> pointing to the <ds:SignatureValue> element of the <ds:Signature> included in the parsed document obtained from b.0.

b.1 The document containing the signature is removed from the set of unprocessed documents, so it's not considered in the rest of the process.

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

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

4.1.5.2 Basic Processing for CMS Signatures

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

1

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

b The server hashes the signature value of the countersigned signature

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

2

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

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

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

4.1.6 Optional Input <ParallelSignature>

This element allows the client to request the creation of a CMS `SignerInfo` over an existing `SignedData` including its encapsulated content. The `SignerInfo` is included in the existing `SignedData` and returned normally in the `<dss:SignatureObject>` of the `<dss:SignResponse>`. The encapsulated content type MUST be `id-data`.

The `<dss:Document>` element within `<dss:InputDocuments>` MUST only contain a `<dss:Base64Data>` including the signature in order to create the parallel signature.

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

4.1.6.1 Basic Processing

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

1

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

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

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

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

4.2 Result Codes

<dss:ResultMajor>	<dss:ResultMinor>	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 <xadp:SignatureForm> element and the one

		included in the signature policy referenced or included in the <SignaturePolicy> are incompatible.
RequesterError	SignatureFormsNotSupported	The client has selected a <dss:SignatureType> that does not support signature forms.
RequesterError	CommitmentNotFound	The selected commitment cannot be found by the server.

501
502

5 Profile of Verifying Protocol

5.1 Optional Inputs and Outputs

5.1.1 Optional Input and Output <dss:VerificationTime>

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

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

The semantics for the different elements are

- for certificates, this verification time allows the client to request the verification of the status of the certificate when requested.
- for timestamps, this verification time allows the client to request the verification of the status of the timestamp when requested.
- for signatures, different cases arise
    - when the signing time is known and trusted, this parameter MAY have no effect and the signature MUST be verified using the trusted signing time. The server returns the optional output `<dss:SigningTime>` instead of `<dss:VerificationTime>`
    - when the signing time is not known, the server MAY perform the signature verification using this time.

The server MUST return the `<dss:VerificationTime>` when this time differs from the one passed in the input, or when no `<dss:VerificationTime>` is requested as an optional input.

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

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

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

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

- the signature policy included in the request (if any).
- the explicit policy included as a signed attribute at signature production (if any).

- the policy defined as the default policy in the server (if any), applicable when no policy is present in the request or in the signature.
- signature policies (if any) determined to be compatible with the policy included in the request or the signature by the service (signature policy mappings).

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

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 (**overriding the one in the signature**).
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) (**overriding the one in the signature**).

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

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

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

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

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

```
   <xs:element name="SignaturePolicy">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="xades:ObjectIdentifierType">
               <xs:attribute name="allowPolicyMappings" type="xs:boolean"
use="optional" default="false"/>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>
   </xs:element>
```

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

`<SignaturePolicyIssuer>` [Required]

The issuer of the signature policy.

`<SignaturePolicyIdentifier>` [Required]

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

`<SignaturePolicyDigestAlgorithm>` [Required]

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

`<SignaturePolicyDigestValue>` [Required]

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

`<ds:Transforms>` [Optional]

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

```
   <xs:element name="SignaturePolicyInfo">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="SignaturePolicyIssuer" type="xs:string"/>
            <xs:element name="SignaturePolicyIdentifier"
type="xades:ObjectIdentifierType"/>
            <xs:element name="SignaturePolicyDigestAlgorithm"
type="xades:ObjectIdentifierType"/>
            <xs:element name="SignaturePolicyDigestValue"
type="ds:DigestValueType"/>
            <xs:element ref="ds:Transforms" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
```

```
    </xs:element>
```

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

5.1.3 Optional Input <Scheme> / Optional output <SchemeInfo>

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

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

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

<SchemeName> [Required]

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

```
<xs:element name="Scheme">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="SchemeName" type="tsl:InternationalNamesType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

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

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

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

<SchemeName> [Required]

The name of the scheme.

<TSLSequenceNumber> [Required]

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

<TSLDigestAlgorithm> [Required]

The algorithm used to digest the TSL object.

627 <TSLDigestValue> [Required]

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

629

```
630    <xs:element name="SchemeInfo">
631       <xs:complexType>
632          <xs:sequence>
633             <xs:element name="SchemeName" type="tsl:InternationalNamesType"/>
634             <xs:element name="TSLSequenceNumber" type="xs:integer"/>
635             <xs:element name="TSLDigestAlgorithm"
636 type="xades:ObjectIdentifierType"/>
637             <xs:element name="TSLDigestValue" type="ds:DigestValueType"/>
638          </xs:sequence>
639       </xs:complexType>
640    </xs:element>
```

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

642 5.1.4 Optional Input <X509CertificateValidationOptions>

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

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

647 5.1.5 Optional Input <dss:ReturnUpdatedSignature> / Optional Output
648 <dss:UpdatedSignature>

649 The server MUST refuse to update the signature, using the minor code
650 SignaturePropertiesNotSupported when the signature type requested is not **[CAdES]** or
651 **[XAdES]**.

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

653 5.1.6 Optional Input <RequireQualifiedCertificate>

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

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

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

660

661

662

663

664

665 ## 5.2 Result Codes

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

6 Identifiers

6.1 Signature Properties Identifiers

6.1.1 Signed Properties

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

Identifier	[XAdES-DSS] Signature Property	[CAdES-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

NOTES:

- The Identifiers MUST be prefixed by `urn:oasis:names:tc:dss:1.0:profiles:XAdES:`.
- The server MUST decide between `SigningCertificate` and `OtherSigningCertificate` using the criteria defined in **[CAdES]**.

6.1.2 Unsigned Properties

Identifier	[XAdES] Signature Property	[CAdES] 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

NOTES:

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

6.2 Signature Form Identifiers

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

7 References

7.1 Normative

[TO BE DONE]

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.

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

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

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

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

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

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

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

Appendix C. Message Authentication using X509 Certificates

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

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

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

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

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

Processing rules in the server MUST include the following checks

- check the cryptographic validity of the signature and its coverage

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

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

Appendix D. Client Authentication using SAML Assertions

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

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

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

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

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

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

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

Processing rules in the server MUST include the following checks

- check the cryptographic validity of the assertion

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

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

Appendix E. Client Authentication using different password-based schemes

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

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

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

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

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

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

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

Appendix F. Usage of Signature Policies in Signature Creation and Verification

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

The creation and verification of signatures will only work when creating / verifying **[XAdES]** or **[CAdES]** signatures.

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

When dealing with verification, two cases arise

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

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

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

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

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

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

```
<xs:complexType name="AttributeType">
    <xs:sequence>
        <xs:element ref="saml20:AttributeValue" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="NameFormat" type="xs:anyURI" use="optional"/>
    <xs:attribute name="FriendlyName" type="xs:string" use="optional"/>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<element name="AttributeValue" type="xs:anyType" nillable="true"/>
```

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

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

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

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.

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

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

The attributes defined in this profile to be used with `<ReturnX509CertificateInfo>` are defined below. All the attributes defined below MUST be prefixed with `urn:oasis:names:tc:dss:1.0:profiles:XSS:certificateAttributes`.

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 **[RFC2253]**.
SubjectDistinguishedName	xs:string	The subject distinguished name, formatted as described in **[RFC2253]**.
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.
Extension:XXX	xs:base64Binary	The ASN.1 value, DER-Encoded of the extension XXX. Valid extension names can be found in **[RFC3280]**.

889
890

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.

Appendix I. 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.