# ETSI TS 101 903 V1.3.2 (2006-03)

*Technical Specification*

**XML Advanced Electronic Signatures (XAdES)**

Reference

RTS/ESI-000034

Keywords

e-commerce, electronic signature, security

*ETSI*

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

# Contents

## Intellectual Property Rights

## Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Electronic Signatures and Infrastructures (ESI).

## Introduction

Electronic commerce is emerging as the future way of doing business between companies across local, wide area and global networks. Trust in this way of doing business is essential for the success and continued development of electronic commerce. It is therefore important that companies using this electronic means of doing business have suitable security controls and mechanisms in place to protect their transactions and to ensure trust and confidence with their business partners. In this respect the electronic signature is an important security component that can be used to protect information and provide trust in electronic business.

The European Directive on a community framework for Electronic Signatures (also denoted as "the Directive" or the "European Directive" in the rest of the present document) defines an electronic signature as: "data in electronic form which is attached to or logically associated with other electronic data and which serves as a method of authentication".

The present document is intended to cover electronic signatures for various types of transactions, including business transactions (e.g. purchase requisition, contract, and invoice applications). Thus the present document can be used for any transaction between an individual and a company, between two companies, between an individual and a governmental body, etc. The present document is independent of any environment. It can be applied to any environment e.g. smart cards, GSM SIM cards, special programs for electronic signatures, etc.

The ETSI standard TS 101 733 [1] defines formats for advanced electronic signatures that remain valid over long periods, are compliant with the European Directive and incorporate additional useful information in common use cases (like indication of the commitment got by the signature production). Currently, it uses Abstract Syntax Notation 1 (ASN.1) and is based on the structure defined in RFC 3369 [2] (in the present document the signatures aligned with this RFC will be denoted as CMS signatures).

TS 101 733 [1]:

- Defines new ASN.1 types able to contain information for qualifying the CMS signatures so that they fulfil the aforementioned requirements.

- Specifies how this qualifying information must be incorporated to the CMS signatures.

Currently, the IETF W3C XML-Signature Working Group has developed a syntax for XML signatures: "XML-Signature Core Syntax and Processing" [3] (denoted as XMLDSIG in the present document). This syntax provides a basic functionality for digitally signing several data objects at the same time. It also provides basic means to incorporate any kind of needed qualifying information.

The present document:

- specifies XML schema [5] definitions for new XML types that can be used to generate properties that further qualify XMLDSIG signatures with information able to fulfil a number of common requirements such as the long term validity of the signature by usage of time-stamps, etc.;
- defines mechanisms for incorporating the aforementioned qualifying information;
- specifies formats for XML advanced electronic signatures that, by using the specified new XML types, remain valid over long periods and incorporate additional useful information in common use cases. These signatures will be built on XMLDSIG by addition of these properties as specified in [3], using the `ds:Object` XML element defined there (here, as for the rest of the document, `ds` has been used as the prefix denoting the namespace defined in [3]. Its value is defined in clause 4);
- defines a set of conformance requirements to claim endorsement to the present document.

The present document specifies two main types of properties: signed properties and unsigned properties. The first ones are additional data objects that are also secured by the signature produced by the signer on the `ds:SignedInfo` element, which implies that the signer gets these data objects, computes a hash for all of them and generates the corresponding `ds:Reference` element. The unsigned properties are data objects added by the signer, by the verifier or by other parties after the production of the signature. They are not secured by the signature in the `ds:Signature` element (the one computed by the signer); however they can be actually signed by other parties (time-stamps, countersignatures, certificates and CRLs are also signed data objects).

The XML advanced electronic signatures defined in the present document will be built by incorporating to the XML signatures as defined in [3] XMLDSIG one new `ds:Object` XML element containing the additional qualifying information.

# Editorial conventions

As it has been anticipated in the former clause, throughout the rest of the document the term XMLDSIG will refer to XML signatures with basic functionality, i.e. to XML signatures that do not incorporate the qualifying information on the signature, the signer or the signed data object(s) specified in the present document.

Throughout the rest of the document the terms "qualifying information", "properties" or "qualifying properties" will be used to refer to the information added to the XMLDSIG to get an XML advanced electronic signature as specified in the European Directive and with long term validity.

For the present document the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in the present document are to be interpreted as described in RFC 2119 [10].

# 1 Scope

The present document defines XML formats for advanced electronic signatures that remain valid over long periods, are compliant with the European Directive and incorporate additional useful information in common uses cases. This includes evidence as to its validity even if the signer or verifying party later attempts to deny (repudiates) the validity of the signature.

The present document is based on the use of public key cryptography to produce digital signatures, supported by public key certificates.

The present document uses a signature policy, implicitly or explicitly referenced by the signer, as one possible basis for establishing the validity of an electronic signature.

The present document uses time-stamps or trusted records (e.g. time-marks) to prove the validity of a signature long after the normal lifetime of critical elements of an electronic signature and to support non-repudiation. It also specifies the optional use of additional time-stamps to provide very long-term protection against key compromise or weakened algorithms.

The present document then, specifies the use of the corresponding trusted service providers (e.g. time-stamping authorities), and the data that needs to be archived (e.g. cross certificates and revocation lists). An advanced electronic signature aligned with the present document can, in consequence, be used for arbitration in case of a dispute between the signer and verifier, which may occur at some later time, even years later.

The present document builds on the standards for Electronic Signatures defined in:

- IETF W3C: "XML-Signature Syntax and Processing" [3];
- TS 101 733: "Electronic Signature Formats" [1];
- ITU-T Recommendation X.509: "Information technology - Open Systems Interconnection - The Directory: Authentication framework" [6];
- RFC 3261: "Internet X.509 Public Key Infrastructure Time-Stamp protocol (TSP)" [13].

NOTE: See clause 2 for a full set of references.

The present document, being built on the framework defined in [3] makes use of the terms defined there. Some of the definitions in [3] are repeated in the present document for the sake of completeness.

The present document:

- shows a taxonomy of the qualifying information (properties) that have to be present in an electronic signature to remain valid over long periods, to satisfy common use cases requirements, and to be compliant with the European Directive;
- specifies XML schema definitions for new elements able to carry or to refer to the aforementioned properties;
- specifies two ways for incorporating the qualifying information to XMLDSIG, namely either by direct incorporation of the qualifying information or using references to such information. Both ways make use of mechanisms defined in XMLDSIG.

Clause 2 in the present document contains references to relevant documents and standards.

Clause 4 gives an overview of the various types of advanced electronic signatures defined in the present document.

Clause 5 contains the namespace specification for the XML schema definitions appearing in the present document.

Clause 6 describes how the qualifying information is added to XMLDSIG.

Clause 7 contains the details (including schema definitions) of the elements where the qualifying information is included.

Clause 8 specifies conformance requirements for claiming endorsement to the present document.

Annex A is normative. It contains definitions for relevant concepts used throughout the present document.

Annex B is informative. It defines extended formats of advanced electronic signatures that include validation data and time-stamps for archival.

Annex C is informative. It presents details on some concepts used in the current specification.

Annex D is normative. It contains the whole set of schema definitions for the elements defined in the present document.

Annex E is informative. It contains the non normative DTD corresponding to the aforementioned schema.

Annex F is informative. It shows examples of how to incorporate qualifying information leading to the XML Advanced Electronic Signatures.

Annex G is informative. It presents certain technical rules that verifiers should take into account when verifying XAdES signatures.

Annex H contains bibliography.

# 2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication and/or edition number or version number) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

[1] ETSI TS 101 733: "Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES)".

[2] IETF RFC 3852 (obsoletes RFC 3369): "Cryptographic Message Syntax (CMS)".

[3] W3C/IETF Recommendation: "XML-Signature Syntax and Processing".

[4] W3C Recommendation part 1: "XML Schema Part 1: Structures".

[5] W3C Recommendation part 2: "XML Schema Part 2: Datatypes".

[6] ITU-T Recommendation X.509: "Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks".

[7] ETSI TS 101 861: "Time stamping profile".

[8] W3C Recommendation: "Extensible Markup Language (XML) 1.0".

[9] IETF RFC 2560: "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP".

[10] IETF RFC 2119: "Key words for use in RFCs to Indicate Requirement Levels".

[11] IETF RFC 3161: "Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP)".

[12] ETSI TR 102 038: "TC Security - Electronic Signatures and Infrastructures (ESI); XML format for signature policies".

[13] IETF RFC 3261: "Internet X.509 Public Key Infrastructure Time-Stamp protocol".

[14] IETF RFC 3280: "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile".

# 3 Definitions and abbreviations

## 3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

**arbitrator:** entity that arbitrates in disputes between a signer and a verifier

**attributes authorities:** provide users with attributes linked to public key certificates

**Certification Authorities (CA):** provide users with public key certificates

**registration authorities:** allow the identification and registration of entities before a CA generates certificates

**repository authorities:** publish CRLs issued by CAs, signature policies issued by signature policy issuers and optionally public key certificates

**signature policy issuers:** define the technical and procedural requirements for electronic signature creation and validation, in order to meet a particular business need

**signer:** entity that creates the electronic signature

NOTE: When the signer digitally signs data object(s) using the prescribed format, this represents a commitment on behalf of the signing entity to the data object(s) being signed.

**Time-Stamping Authorities (TSA):** attest that some data object was formed before a given trusted time

**time-marking authorities:** record that some data was formed before a given trusted time

**Trusted Service Providers** (TSPs)**:** one or more entities that help to build trust relationships between the signer and verifier

NOTE: They support the signer and verifier by means of supporting services including user certificates, cross-certificates, time-stamping tokens, CRLs, AARLs, OCSP responses.

**verifier:** entity that verifies the electronic signature

## 3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

| | |
|---|---|
| AC | Attribute Certificate |
| CA | Certification Authority |
| CMS | Cryptographic Message Syntax |
| CRL | Certificate Revocation List |
| DTD | Document Type Definition |
| ES | Electronic Signature |
| HTTP | Hyper Text Transfer Protocol |
| OCSP | Online Certificate Status Protocol |
| OID | Object IDentifier |
| PKC | Public Key Certificate |
| TSA | Time-Stamping Authorities |
| TSP | Trusted Service Providers |
| TSU | Time Stamping Unit |
| URI | Uniform Resource Identifier |
| URN | Uniform Resource Name |
| XAdES | XML Advanced Electronic Signature |
| XAdES-A | XAdES Archiving validation data |
| XAdES-BES | XAdES Basic Electronic Signature |
| XAdES-C | XAdES Complete validation data |
| XAdES-EPES | XAdES Explicit Policy based Electronic Signature |
| XAdES-T | XAdES with Time-stamp |

| | |
|---|---|
| XAdES-X | XAdES eXtended validation data |
| XML | eXtensible Markup Language |
| XMLDSIG | eXtensible Markup Language Digital SIGnature |
| XSL | eXtensible Stylesheet Language |
| XSLT | eXtensible Stylesheet Language Transformations |

# 4 Overview

## 4.1 Major Parties

The following are the major parties involved in a business transaction supported by electronic signatures as defined in the present document:

- the Signer;
- the Verifier;
- Trusted Service Providers (TSP);
- the Arbitrator.

The following TSPs are used to support the functions defined in the present document:

- Certification Authorities;
- Registration Authorities;
- Repository Authorities (e.g. a directory);
- Time-Stamping Authorities;
- Time-Marking Authorities;
- Signature Policy Issuers;
- Attribute Authorities.

## 4.2 Signatures policies

The present document includes the concept of signature policies that can be used to establish technical consistency when validating electronic signatures. When a comprehensive signature policy used by the verifier is either explicitly indicated by the signer or implied by the data being signed, then a consistent result can be obtained when validating an electronic signature. When the signature policy being used by the verifier is neither indicated by the signer nor can be derived from other data, or the signature policy is incomplete then verifiers, including arbitrators, may obtain different results when validating an electronic signature. Therefore, comprehensive signature policies that ensure consistency of signature validation are recommended from both the signers and verifiers point of view.

Specification of the contents of signature policies is outside the scope of the current document. Further information on signature policies is provided in TR 102 038 [12].

## 4.3 Signature properties and signature forms

The present document defines a set of signature properties that MAY be combined to obtain electronic signature forms providing satisfaction of different requirements. Below follows a short overview of the properties:

- `SignaturePolicyIdentifier`. This property contains information being an **unambiguous way for identifying the signature policy** under which the electronic signature has been produced. This will ensure that the verifier will be able to use the same signature policy during the verification process. A signature policy is useful to clarify the precise role and commitments that the signer intends to assume with respect to the signed data object, and to avoid claims by the verifier that a different signature policy was implied by the signer. Details on this property can be found in clause 7.2.3.

- Validation data properties. The present document defines a number of XML types able to contain both validation data (certificate chains, CRLs, OCSP responses, etc) and references to them (identifiers of certificates, CRLs, OCSP responses, etc). Properties of these types allow to incorporate all material used for validation into the signature. They can be jointly used with time-stamp properties to provide long term validity. Below follows the list of properties:

  - `CompleteCertificateRefs`. It contains references to the CA certificates used to validate the signature. Details on this property can be found in clause 7.4.1.

  - `CompleteRevocationRefs`. It contains references to the full set of revocation information used for the verification of the electronic signature. Details on this property can be found in clause 7.4.2.

  - `AttributeCertficateRefs`. It contains references to the full set of Attribute Authorities certificates that have been used to validate the attribute certificate. Details on this property can be found in clause 7.4.3.

  - `AttributeRevocationRefs`. It contains references to the full set of references to the revocation data that have been used in the validation of the attribute certificate(s) present in the signature. Details on this property can be found in clause 7.4.4.

  - `CertificateValues`. It contains the values of certificates used to validate the signature. Details on this property can be found in clause 7.6.1.

  - `RevocationValues`. It contains the full set of revocation information used for the verification of the electronic signature. Details on this property can be found in clause 7.6.2.

  - `AttrAuthoritiesCertValues`. It contains values of the Attribute Authorities certificates that have been used to validate the attribute certificate when present in the signature. Details on this property can be found in clause 7.6.3.

  - `AttributeRevocationValues`. It contains the full set of revocation data that have been used to validate the attribute certificate when present in the signature. Details on this property can be found in clause 7.6.4.

- `Time`-stamp token container properties. The present document defines an abstract and two concrete XML types (Generic`TimeStampType`, `XAdESTimeStampType` and `OtherTimeStampType`) for allowing the inclusion of time-stamp tokens in a XMLDSIG signature. These types are defined in clause 7.1.4. The present document uses `XAdESTimeStampType` for defining several time-stamp token container properties, each one containing one or more time-stamp tokens covering different parts of the signature (common elements defined in XMLDSIG, validation data, qualifying properties, etc). Below follows the list:

  - `SignatureTimeStamp`. Each time-stamp token within this property covers the digital signature value element. Details on this property can be found in clause 7.3.

  - `AllDataObjectsTimeStamp`. Each time-stamp token within this property covers all the signed data objects. Details on this property can be found in clause 7.2.9.

  - `IndividualDataObjectsTimeStamp`. Each time-stamp token within this property covers selected signed data objects. Details on this property can be found in clause 7.2.10.

  - `SigAndRefsTimeStamp`. Each time-stamp token within this property covers the signature and references to validation data. Details on this property can be found in clause 7.5.1.

  - `RefsOnlyTimeStamp`. Each time-stamp token within this property covers only references to validation data. Details on this property can be found in clause 7.5.2.
  - `ArchiveTimeStamp`. Each time-stamp token within this property covers signature and other properties required for providing long-term validity. Details on this property can be found in clause 7.7.

- Other properties. The present document defines a number of additional properties that can be useful in a wide range of environments, namely:
  - `SigningCertificate`. This property contains an unambiguous **reference to the signer's certificate**, formed by its identifier and the digest value of the certificate. Its usage is particularly important when a signer holds a number of different certificates containing the same public key, to avoid claims by a verifier that the signature implies another certificate with different semantics. This is also important when the signer holds different certificates containing different public keys in order to provide the verifier with the correct signature verification data. Finally, it is also important in case the issuing key of the CA providing the certificate would be compromised. Details on this property can be found in clause 7.2.2.
  - `SigningTime`. This property contains the time at which the signer claims to have performed the signing process. Details on this property can be found in clause 7.2.1.
  - `DataObjectFormat`. This property identifies the format of a signed data object (when electronic signatures are not exchanged in a restricted context) to enable the presentation to the verifier or use by the verifier (text, sound or video) in exactly the same way as intended by the signer. Details on this property can be found in clause 7.2.5.
  - `CommitmentTypeIndication`. This property identifies the commitment undertaken by the signer in signing (a) signed data object(s) in the context of the selected signature policy (when an explicit commitment is being used). This will be required where a signature policy specifies more than a single commitment type, each of which might have different legal interpretations of the intent of the signature (e.g. proof of origin, proof of receipt, proof of creation, etc.). Details on this property can be found in clause 7.2.6.
  - `SignatureProductionPlace`. This property contains the indication of the purported place where the signer claims to have produced the signature. Details on this property can be found in clause 7.2.7.
  - `SignerRole`. This property contains claimed or certified roles assumed by the signer in creating the signature. Details on this property can be found in clause 7.2.8.
  - `CounterSignature`. This property contains signature(s) produced on the signature. Details on this property can be found in clause 7.2.4.

The aforementioned properties are defined in the normative part of the present document. They can be combined to generate different electronic signature forms. Some of them are defined in clause 4.4 of its normative part. Additional extended forms are defined in the informative annex B. Clause 8 specifies conformance requirements for claiming endorsement to the present technical specification.

## 4.4 Electronic signature forms

The current clause specifies four forms of XML advanced electronic signatures, namely the **Basic Electronic Signature** (XAdES-BES), the **Explicit Policy based Electronic Signature** (XAdES-EPES), and the **Electronic Signature with Validation Data** (XAdES-T and XAdES-C). Conformance to the present document mandates the signer to create one of these formats.

The informative annex B defines extended forms of XAdES. Conformance to the present document does not mandate the signer to create any of the forms defined in annex B.

### 4.4.1 Basic electronic signature (XAdES-BES)

A Basic Electronic Signature (XAdES-BES) in accordance with the present document will build on a XMLDSIG by incorporating qualifying properties defined in the present specification. They will be added to XMLDSIG within one `ds:Object` acting as the bag for the whole set of qualifying properties, or by using the mechanism defined in clause 6.3.2 that allows further distribution of the properties.

Some properties defined for building up this form will be covered by the signer's signature (signed qualifying information grouped within one new element, `SignedProperties`, see clause 6.2.1). Other properties will be not covered by the signer's signature (unsigned qualifying information grouped within one new element, `UnsignedProperties`, see clause 6.2.2).

In a XAdES-BES the signature value SHALL be computed in the usual way of XMLDSIG over the data object(s) to be signed **and on the whole set of signed properties when present** (`SignedProperties` element).

For this form it is mandatory to protect the signing certificate with the signature, in one of the two following ways:

- Either incorporating the `SigningCertificate` signed property; or
- Not incorporating the `SigningCertificate` but incorporating the signing certificate within the `ds:KeyInfo` element and signing it.

A XAdES-BES signature MUST, in consequence, contain at least one of the following elements with the specified contents:

- The `SigningCertificate` signed property. This property MUST contain the reference and the digest value of the signing certificate. It MAY contain references and digests values of other certificates (that MAY form a chain up to the point of trust). In the case of ambiguities identifying the actual signer's certificate the applications SHOULD include the `SigningCertificate` property.
- The `ds:KeyInfo` element. If `SigningCertificate` is present in the signature, no restrictions apply to this element. If `SigningCertificate` element is not present in the signature, then the following restrictions apply:
  - the `ds:KeyInfo` element MUST include a `ds:X509Data` containing the signing certificate;
  - the `ds:KeyInfo` element also MAY contain other certificates forming a chain that MAY reach the point of trust;
  - the `ds:SignedInfo` element MUST contain a `ds:Reference` element referencing `ds:KeyInfo`, so that the latter is included in the signature value computation. In this way, the signing certificate is secured by the signature.

By incorporating one of these elements, XAdES-BES prevents the simple substitution of the signer's certificate (see clause 7.2.2).

A XAdES-BES signature MAY also contain the following properties:

- the `SigningTime` signed property;
- the `DataObjectFormat` signed property;
- the `CommitmentTypeIndication` signed property;
- the `SignerRole` signed property;
- the `SignatureProductionPlace` signed property;
- one or more `IndividualDataObjectsTimeStamp` or `AllDataObjectTimeStamp` signed properties;
- one or more `CounterSignature` unsigned properties.

Below follows the structure of the XAdES-BES built **by direct incorporation** of the qualifying information in the corresponding new XML elements to the XMLDSIG (see clause 6.3.1 for further details). In the example "?" denotes zero or one occurrence; "+" denotes one or more occurrences; and "*" denotes zero or more occurrences.

The XML schema definition in clause 5 defines the prefix "`ds`" for all the XML elements already defined in XMLDSIG, and states that the default namespace is the one defined for the present document. In consequence, in the example of this clause, the elements already defined in XMLDSIG appear with the prefix "`ds`", whereas the new XML elements defined in the present document appear without prefix.

```
                              XMLDSIG
                                   |
<ds:Signature ID?>- - - - - - - - -+- - - - -+
  <ds:SignedInfo>                  |         |
    <ds:CanonicalizationMethod/>   |         |
    <ds:SignatureMethod/>          |         |
    (<ds:Reference URI? >          |         |
      (<ds:Transforms>)?           |         |
      <ds:DigestMethod/>           |         |
      <ds:DigestValue/>            |         |
    </ds:Reference>)+              |         |
  </ds:SignedInfo>                 |         |
  <ds:SignatureValue/>             |         |
  (<ds:KeyInfo>)?- - - - - - - - - +         |
                                             |
  <ds:Object>                                |
                                             |
    <QualifyingProperties>                   |
                                             |
      <SignedProperties>                     |
                                             |
        <SignedSignatureProperties>          |
          (SigningTime)?                     |
          (SigningCertificate)?              |
          (SignatureProductionPlace)?        |
          (SignerRole)?                      |
        </SignedSignatureProperties>         |
                                             |
        <SignedDataObjectProperties>         |
          (DataObjectFormat)*                |
          (CommitmentTypeIndication)*        |
          (AllDataObjectsTimeStamp)*         |
          (IndividualDataObjectsTimeStamp)*  |
        </SignedDataObjectProperties>        |
                                             |
      </SignedProperties>                    |
                                             |
      <UnsignedProperties>                   |
                                             |
        <UnsignedSignatureProperties>        |
          (CounterSignature)*                |
        </UnsignedSignatureProperties>       |
                                             |
      </UnsignedProperties>                  |
                                             |
    </QualifyingProperties>                  |
                                             |
  </ds:Object>                               |
                                             |
</ds:Signature>- - - - - - - - - - - - - - - +
                                             |
                                        XAdES-BES
```

Other XMLDSIG `ds:Object` elements with different contents MAY be added within the structure shown above to satisfy requirements other than the ones expressed in the present document. This also applies to the rest of the examples of structures of XAdES forms shown in this clause.

The signer's conformance requirements of a XAdES-BES are defined in clause 8.1.

NOTE: The XAdES-BES is the minimum format for an electronic signature to be generated by the signer. On its own, it does not provide enough information for it to be verified in the longer term. For example, revocation information issued by the relevant certificate status information issuer needs to be available for long term validation (see clause 4.4.3).

The XAdES-BES satisfies the legal requirements for electronic signatures as defined in the European Directive on electronic signatures. It provides basic authentication and integrity protection.

The semantics of the signed data of a XAdES-BES or its context may implicitly indicate a signature policy to the verifier.

## 4.4.2 Explicit policy electronic signatures (XAdES-EPES)

An **Explicit Policy based Electronic Signature** (XAdES-EPES) form in accordance with the present document, extends the definition of an electronic signature to conform to the identified signature policy. A XAdES-EPES builds up on a XMLDSIG or XAdES-BES forms by incorporating the `SignaturePolicyIdentifier` element. This signed property indicates that a signature policy MUST be used for signature validation. It MAY explicitly identify the signature policy. Other properties may be required by the mandated policy.

Clause 7.2.3 provides details on the specification of `SignaturePolicyIdentifier` property. Specification of the actual signature policies is outside the scope of the current document. Further information on signature policies is provided in TR 102 038 [12].

The structure of the XAdES-EPES (created **by direct incorporation** of the qualifying information to a XAdES-BES form) is illustrated below.

```
                              XMLDSIG
                                   |
<ds:Signature ID?>- - - - - - - - -+- - - - -+
  <ds:SignedInfo>                  |         |
    <ds:CanonicalizationMethod/>   |         |
    <ds:SignatureMethod/>          |         |
    (<ds:Reference URI? >          |         |
      (<ds:Transforms>)?           |         |
      <ds:DigestMethod/>           |         |
      <ds:DigestValue/>            |         |
    </ds:Reference>)+              |         |
  </ds:SignedInfo>                 |         |
  <ds:SignatureValue/>             |         |
  (<ds:KeyInfo>)?- - - - - - - - - +         |
                                             |
  <ds:Object>                                |
                                             |
    <QualifyingProperties>                   |
                                             |
      <SignedProperties>                     |
                                             |
        <SignedSignatureProperties>          |
          (SigningTime)?                     |
          (SigningCertificate)?              |
          (SignaturePolicyIdentifier)        |
          (SignatureProductionPlace)?        |
          (SignerRole)?                      |
        </SignedSignatureProperties>         |
                                             |
        <SignedDataObjectProperties>         |
          (DataObjectFormat)*                |
          (CommitmentTypeIndication)*        |
          (AllDataObjectsTimeStamp)*         |
          (IndividualDataObjectsTimeStamp)*  |
        </SignedDataObjectProperties>        |
                                             |
      </SignedProperties>                    |
                                             |
      <UnsignedProperties>                   |
                                             |
        <UnsignedSignatureProperties>        |
          (CounterSignature)*                |
        </UnsignedSignatureProperties>       |
                                             |
      </UnsignedProperties>                  |
                                             |
    </QualifyingProperties>                  |
                                             |
  </ds:Object>                               |
                                             |
</ds:Signature>- - - - - - - - - - - - - - - +
                                             |
                                        XAdES-EPES
```

The signer's conformance requirements of XAdES-EPES are defined in clause 8.2.

### 4.4.3 Electronic signature formats with validation data

Validation of an electronic signature in accordance with the present document requires additional data needed to validate the electronic signature. This additional data is called **validation data**; and includes:

- Public Key Certificates (PKCs) and Attributes Certificates (ACs);
- revocation status information for each PKC and AC;
- trusted time-stamps applied to the digital signature or a time-mark that shall be available in an audit log;
- when appropriate, the details of a signature policy to be used to verify the electronic signature.

The **validation data** may be collected by the signer and/or the verifier. When the signature policy identifier is present, it shall meet the requirements of the signature policy. Validation data includes CA certificates as well as revocation status information in the form of Certificate Revocation Lists (CRLs) or certificate status information (OCSP) provided by an on-line service. Validation data also includes evidence that the signature was created before a particular point in time. This may be either a time-stamp token or time-mark.

The present document defines properties able to contain validation data. Clauses below summarize some signature formats that incorporate them and their most relevant characteristics.

#### 4.4.3.1 Electronic signature with time (XAdES-T)

XML Advanced Electronic Signature with Time (XAdES-T) is a signature for which there exists a trusted time associated to the signature. The trusted time may be provided by two different means:

- the `SignatureTimeStamp` as an unsigned property added to the electronic signature;
- a time mark of the electronic signature provided by a trusted service provider.

A time-mark provided by a Trusted Service would have similar effect to the `SignatureTimeStamp` property but in this case no property is added to the electronic signature as it is the responsibility of the TSP to provide evidence of a time mark when required to do so. The management of time marks is outside the scope of the current document.

Trusted time provides the initial steps towards providing long term validity.

Below follows the structure of a XAdES-T form built on a XAdES-BES or a XAdES-EPES, by direct incorporation of a time-stamp token within the `SignatureTimeStamp` element. A XAdES-T form based on time-marks MAY exist without such an element.

```
                              XMLDISG
                                  |
<ds:Signature ID?>- - - - - - - - +- - - - +- - - +
  <ds:SignedInfo>                 |        |      |
    <ds:CanonicalizationMethod/>  |        |      |
    <ds:SignatureMethod/>         |        |      |
    (<ds:Reference URI? >         |        |      |
      (<ds:Transforms>)?          |        |      |
      <ds:DigestMethod/>          |        |      |
      <ds:DigestValue/>           |        |      |
    </ds:Reference>)+             |        |      |
  </ds:SignedInfo>                |        |      |
  <ds:SignatureValue/>            |        |      |
  (<ds:KeyInfo>)? - - - - - - - - +        |      |
                                           |      |
  <ds:Object>                              |      |
                                           |      |
    <QualifyingProperties>                 |      |
                                           |      |
      <SignedProperties>                   |      |
                                           |      |
        <SignedSignatureProperties>        |      |
          (SigningTime)?                   |      |
          (SigningCertificate)?            |      |
          (SignaturePolicyIdentifier)?     |      |
          (SignatureProductionPlace)?      |      |
          (SignerRole)?                    |      |
        </SignedSignatureProperties>       |      |
                                           |      |
        <SignedDataObjectProperties>       |      |
          (DataObjectFormat)*              |      |
          (CommitmentTypeIndication)*      |      |
          (AllDataObjectsTimeStamp)*       |      |
          (IndividualDataObjectsTimeStamp)*|      |
        </SignedDataObjectProperties>      |      |
                                           |      |
      </SignedProperties>                  |      |
                                           |      |
      <UnsignedProperties>                 |      |
                                           |      |
        <UnsignedSignatureProperties>      |      |
          (CounterSignature)*- - - - - - - +      |
          (SignatureTimeStamp)+                   |
        </UnsignedSignatureProperties>- - -+      |
                                           |      |
      </UnsignedProperties>                |      |
                                           |      |
    </QualifyingProperties>                |      |
                                           |      |
  </ds:Object>                             |      |
                                           |      |
</ds:Signature>- - - - - - - - - - - - - - +- - - +
                                           |      |
                                XAdES-BES(-EPES)  |
                                                  |
                                            XAdES-T
```

### 4.4.3.2 Electronic signature with complete validation data references (XAdES-C)

XML Advanced Electronic Signature with Complete validation data references (XAdES-C) in accordance with the present document adds to the XAdES-T the `CompleteCertificateRefs` and `CompleteRevocationRefs` unsigned properties as defined by the present document. If attribute certificates appear in the signature, then XAdES-C also incorporates the `AttributeCertificateRefs` and the `AttributeRevocationRefs` elements.

`CompleteCertificateRefs` element contains a sequence of references to the full set of CA certificates that have been used to validate the electronic signature up to (but not including) the signing certificate.

`CompleteRevocationRefs` element contains a full set of references to the revocation data that have been used in the validation of the signer and CA certificates.

`AttributeCertificateRefs` and `AttributeRevocationRefs` elements contain references to the full set of Attribute Authorities certificates and references to the full set of revocation data that have been used in the validation of the attribute certificates present in the signature, respectively.

Storing the references allows the values of the certification path and revocation data to be stored elsewhere, reducing the size of a stored electronic signature format.

Below follows the structure for XAdES-C built by direct incorporation of properties on a XAdES-T containing the `SignatureTimeStamp` signed property. A XAdES-C form based on time-marks MAY exist without such element.

```
                              XMLDISG
                                  |
<ds:Signature ID?>- - - - - - - - +- - - - - - +-+-+
  <ds:SignedInfo>                 |            | | |
    <ds:CanonicalizationMethod/>  |            | | |
    <ds:SignatureMethod/>         |            | | |
   (<ds:Reference URI? >          |            | | |
      (<ds:Transforms>)?          |            | | |
      <ds:DigestMethod/>          |            | | |
      <ds:DigestValue/>           |            | | |
    </ds:Reference>)+             |            | | |
  </ds:SignedInfo>                |            | | |
  <ds:SignatureValue/>            |            | | |
  (<ds:KeyInfo>)? - - - - - - - - +            | | |
                                               | | |
  <ds:Object>                                  | | |
                                               | | |
    <QualifyingProperties>                     | | |
                                               | | |
      <SignedProperties>                       | | |
                                               | | |
        <SignedSignatureProperties>            | | |
        (SigningTime)?                         | | |
        (SigningCertificate)?                  | | |
        (SignaturePolicyIdentifier)?           | | |
        (SignatureProductionPlace)?            | | |
        (SignerRole)?                          | | |
      </SignedSignatureProperties>             | | |
                                               | | |
      <SignedDataObjectProperties>             | | |
        (DataObjectFormat)*                    | | |
        (CommitmentTypeIndication)*            | | |
        (AllDataObjectsTimeStamp)*             | | |
        (IndividualDataObjectsTimeStamp)*      | | |
      </SignedDataObjectProperties>            | | |
                                               | | |
      </SignedProperties>                      | | |
                                               | | |
      <UnsignedProperties>                     | | |
                                               | | |
        <UnsignedSignatureProperties>          | | |
          (CounterSignature)*- - - - - - - - - + | |
          (SignatureTimeStamp)+- - - - - - - - - + |
          (CompleteCertificateRefs)                |
          (CompleteRevocationRefs)                 |
          (AttributeCertificateRefs)?              |
          (AttributeRevocationRefs)?               |
        </UnsignedSignatureProperties>- - - -  +-+ |
                                               | | |
       </UnsignedProperties>                   | | |
                                               | | |
    </QualifyingProperties>                    | | |
                                               | | |
  </ds:Object>                                 | | |
                                               | | |
</ds:Signature>- - -  - - - - - - - - - - - - -+-+-+
                                               | | |
                                 XAdES-BES(-EPES)| |
                                                 | |
                                           XAdES-T |
                                                   |
                                             XAdES-C
```

NOTE 1: As a minimum, the signer will provide the XAdES-BES or when indicating that the signature conforms to an explicit signing policy the XAdES-EPES.

NOTE 2: To reduce the risk of repudiating signature creation, the trusted time indication needs to be as close as possible to the time the signature was created. The signer or a TSP could provide the XAdES-T. If the signer did not provide it, the verifier SHOULD create the XAdES-T on first receipt of an electronic signature, because the XAdES-T provides independent evidence of the existence of the signature prior to the trusted time indication.

NOTE 3: The XAdES-T trusted time indications MUST be created before a certificate has been revoked or expired.

NOTE 4: The signer or the TSP MAY provide the XAdES-C to minimize this risk and when the signer does not provide the XAdES-C, the verifier SHOULD create the XAdES-C when the required components of revocation and validation data become available. This MAY require a grace period.

NOTE 5: A grace period permits certificate revocation information to propagate through the revocation processes. This period could extend from the time an authorized entity requests certificate revocation, to when relying parties may be expected to have accessed the revocation information (for example, by contractual requirements placed on relying parties). In order to make sure that the certificate was not revoked at the time the signature was time-marked or time-stamped, verifiers SHOULD wait until the end of the grace period. An illustration of a grace period is provided figure 1.

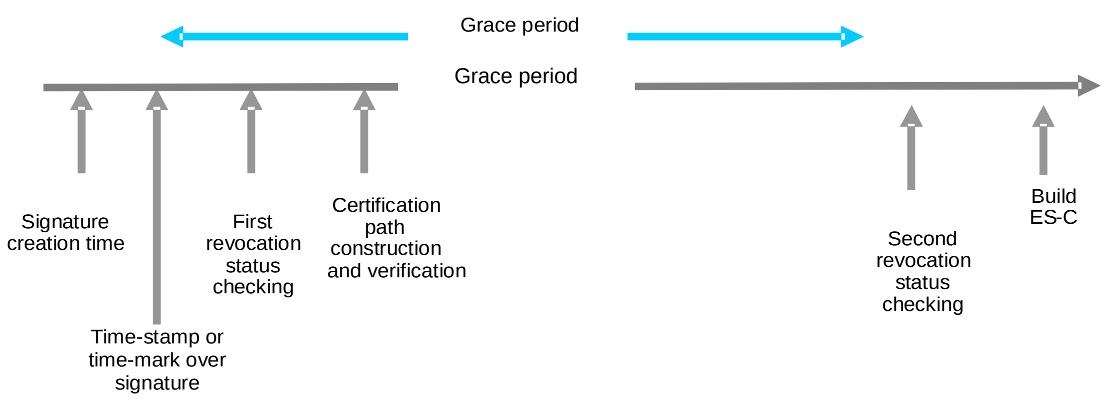

Grace period
Grace period
Signature creation time
Time-stamp or time-mark over signature
First revocation status checking
Certification path construction and verification
Second revocation status checking
Build ES-C

**Figure 1: Illustration of a Grace Period**

The verifier's conformance requirements are defined in clause 8.3 for time stamped XAdES-C and clause 8.4 for time marked XAdES-C signatures.

## 4.5 Validation process

The **Validation Process** validates an electronic signature, the output status of the validation process can be:

- invalid;
- incomplete validation;
- valid.

An **Invalid** response indicates that either the signature format is incorrect or that the digital signature value fails verification (e.g. the integrity check on the digital signature value fails or any of the certificates on which the digital signature verification depends is known to be invalid or revoked).

An **Incomplete Validation** response indicates that the format and digital signature verifications have not failed but there is insufficient information to determine if the electronic signature is valid. For example; all the required certificates are not available or the grace period is not completed. In the case of Incomplete Validation, the electronic signature may be checked again at some later time when additional validation information becomes available. Also, in the case of incomplete validation, additional information may be made available to the application or user, thus allowing the application or user to decide what to do with partially correct electronic signatures.

A **Valid** response indicates that the signature has passed verification and it complies with the signature validation policy.

Informative annex G gives details on technical rules that verifiers should follow for verifying XAdES signatures.

## 4.6 Arbitration

In case of arbitration, a XAdES-C form provides reliable evidence for a valid electronic signature, provided that:

- the arbitrator knows where to retrieve the signer's certificate (if not already present), all the required certificates and CRLs, ACRLs or OCSP responses referenced in the XAdES-C;
- when time-stamping in the XAdES-T is being used, the certificate from the TSU that has issued the time-stamp token in the XAdES-T format is still within its validity period;
- when time-stamping in the XAdES-T is being used, the certificate from the TSU that has issued the time-stamp token in the XAdES-T format is not revoked at the time of arbitration;
- when time-marking in the XAdES-T is being used, a reliable audit trail from the Time-Marking Authority is available for examination regarding the time;
- none of the private keys corresponding to the certificates used to verify the signature chain have ever been compromised;

- the cryptography used at the time the XAdES-C was built has not been broken at the time the arbitration is performed.

If the signature policy can be explicitly or implicitly identified then an arbitrator is able to determine the rules required to validate the electronic signature.

# 5 XML namespace for the present document

The XML namespace URI that MUST be used by implementations of the present document:

http://uri.etsi.org/01903/v1.3.2#

The following namespace declarations apply for the XML Schema definitions throughout the present document.

```
<?xml version="1.0"?>
<schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://uri.etsi.org/01903/v1.3.2#"
  targetNamespace="http://uri.etsi.org/01903/v1.3.2#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  elementFormDefault="qualified">
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd"/>
```

# 6 Syntax overview

This clause introduces the syntax for adding qualifying information to an XML signature.

Clause 6.1 lists a set of technical criteria that has been taken into account for this syntax proposal.

Clause 6.2 specifies an XML element that acts as a container for the qualifying information. Additionally it describes the connection between the XML signature and this container element.

Clause 6.3 shows two ways of incorporating such qualifying information to XMLDSIG.

## 6.1 Technical criteria

The following considerations have been taken into account for the syntax specification for qualifying information on XML signatures:

- The present document specifies how to add qualifying information to an XML signature such that it satisfies both the requirements for an Advanced Electronic Signature according to the European Directive On Electronic Signatures and for remaining valid over long period of time. TS 101 733 [1] identifies all the required information to be added in order to satisfy those requirements. Additionally it defines appropriate data structures for those qualifying properties using ASN.1, that fit for CMS [2] style electronic signatures. The aim of the present document is to specify similar XML qualifying properties that carry such qualifying information and are used to amend XMLDSIG.

- The new XML qualifying properties should not be the result of a stubborn translation process from ASN.1 to XML. This would mean neglecting syntactic differences between CMS [2] and XMLDSIG such as the possible number of signers and multiple signed data objects covered by a single signature, as well as ignoring powerful features of the XML environment such as linking information by using Uniform Resource Identifiers (URI).

- XML Schema [5] has been chosen as the normative language for defining the new XML structures in the present document rather than the DTD vocabulary defined in XML 1.0 [8], since it is namespace aware, allows reuse of existing structures and allows a stricter definition of the allowed contents.

- XML structures that have been defined in related XML standards such as XML Schema [5] and XMLDSIG [3] have been reused where appropriate.

## 6.2 The `QualifyingProperties` element

The `QualifyingProperties` element acts as a container element for all the qualifying information that should be added to an XML signature. The element has the following structure.

```
<xsd:element name="QualifyingProperties" type="QualifyingPropertiesType"/>
<xsd:complexType name="QualifyingPropertiesType">
  <xsd:sequence>
    <xsd:element name="SignedProperties" type="SignedPropertiesType"
      minOccurs="0"/>
    <xsd:element name="UnsignedProperties"
      type="UnsignedPropertiesType"
      minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Target" type="xsd:anyURI" use="required"/>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The qualifying properties are split into properties that are cryptographically bound to (i.e. signed by) the XML signature (`SignedProperties`), and properties that are not cryptographically bound to the XML signature (`UnsignedProperties`). The `SignedProperties` MUST be covered by a `ds:Reference` element of the XML signature.

The mandatory `Target` attribute MUST refer to the `Id` attribute of the corresponding `ds:Signature`. Its value MUST be an URI with a bare-name XPointer fragment. When this element is enveloped by the XAdES signature, its not-fragment part MUST be empty. Otherwise, its not-fragment part MAY NOT be empty.

The optional `Id` attribute can be used to make a reference to the `QualifyingProperties` container.

### 6.2.1 The `SignedProperties` element

The `SignedProperties` element contains a number of properties that are collectively signed by the XMLDSIG signature.

Below follows the schema definition for `SignedProperties` element.

```
<xsd:element name="SignedProperties" type="SignedPropertiesType" />

<xsd:complexType name="SignedPropertiesType">
  <xsd:sequence>
    <xsd:element name="SignedSignatureProperties"
      type="SignedSignaturePropertiesType"/>
      <xsd:element name="SignedDataObjectProperties"
        type="SignedDataObjectPropertiesType" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The `SignedProperties` element MUST contain properties that qualify the XMLDSIG signature itself or the signer. They are included as content of the `SignedSignatureProperties` element.

The `SignedProperties` element MAY also contain properties that qualify some of the signed data objects. These properties appear as content of the `SignedDataObjectProperties` element.

The optional `Id` attribute can be used to make a reference to the `SignedProperties` element.

### 6.2.2 The `UnsignedProperties` element

The `UnsignedProperties` element contains a number of properties that are not signed by the XMLDSIG signature.

```
<xsd:element name="UnsignedProperties" type="UnsignedPropertiesType" />

<xsd:complexType name="UnsignedPropertiesType">
  <xsd:sequence>
    <xsd:element name="UnsignedSignatureProperties"
      type="UnsignedSignaturePropertiesType" minOccurs="0"/>
      <xsd:element name="UnsignedDataObjectProperties"
        type="UnsignedDataObjectPropertiesType" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The `UnsignedProperties` element MAY contain properties that qualify XML signature itself or the signer. They are included as content of the `UnsignedSignatureProperties` element.

The `UnsignedProperties` element MAY also contain properties that qualify some of the signed data objects. These properties appear as content of the `UnsignedDataObjectProperties` element.

The optional `Id` attribute can be used to make a reference to the `UnsignedProperties` element.

### 6.2.3 The `SignedSignatureProperties` element

This element contains properties that qualify the XML signature that has been specified with the `Target` attribute of the `QualifyingProperties` container element.

```
<xsd:element name="SignedSignatureProperties"
  type="SignedSignaturePropertiesType" />

<xsd:complexType name="SignedSignaturePropertiesType">
  <xsd:sequence>
    <xsd:element name="SigningTime" type="xsd:dateTime"
     minOccurs="0"/>
    <xsd:element name="SigningCertificate" type="CertIDListType"
     minOccurs="0"/>
    <xsd:element name="SignaturePolicyIdentifer"
     type="SignaturePolicyIdentifierType" minOccurs="0"/>
    <xsd:element name="SignatureProductionPlace"
     type="SignatureProductionPlaceType"
      minOccurs="0"/>
    <xsd:element name="SignerRole" type="SignerRoleType"
     minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The optional `Id` attribute can be used to make a reference to the `SignedSignatureProperties` element.

The qualifying property `SigningTime` is described in detail in clause 7.2.1, `SigningCertificate` in clause 7.2.2, `SignaturePolicyIdentifier` in clause 7.2.3, `SignatureProductionPlace` in clause 7.2.7, and `SignerRole` in clause 7.2.8.

### 6.2.4 The `SignedDataObjectProperties` element

This element contains properties that qualify some of the signed data objects.

```
<xsd:element name="SignedDataObjectProperties"
  type="SignedDataObjectPropertiesType"/>

<xsd:complexType name="SignedDataObjectPropertiesType">
  <xsd:sequence>
    <xsd:element name="DataObjectFormat" type="DataObjectFormatType"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="CommitmentTypeIndication"
      type="CommitmentTypeIndicationType" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="AllDataObjectsTimeStamp" type="XAdESTimeStampType"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="IndividualDataObjectsTimeStamp"
      type="XAdESTimeStampType"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The optional `Id` attribute can be used to make a reference to the `SignedDataObjectProperties` element.

The qualifying property `AllDataObjectsTimeStamp` is described in detail in clause 7.2.9, `IndividualDataObjectsTimeStamp` in clause 7.2.10, `DataObjectFormat` in clause 7.2.5, and `CommitmentTypeIndication` in clause 7.2.6.

All these properties qualify the signed data object after all the required transforms have been made.

### 6.2.5 The `UnsignedSignatureProperties` element

This element contains properties that qualify the XML signature that has been specified with the `Target` attribute of the `QualifyingProperties` container element. The content of this element is not covered by the XML signature.

```
<xsd:element name="UnsignedSignatureProperties"
  type="UnsignedSignaturePropertiesType"/>

<xsd:complexType name="UnsignedSignaturePropertiesType">
  <xsd:choice maxOccurs="unbounded">
    <xsd:element name="CounterSignature" type="CounterSignatureType" />
    <xsd:element name="SignatureTimeStamp" type="XAdESTimeStampType/>
    <xsd:element name="CompleteCertificateRefs"
      type="CompleteCertificateRefsType"/>
    <xsd:element name="CompleteRevocationRefs"
      type="CompleteRevocationRefsType"/>
    <xsd:element name="AttributeCertificateRefs"
      type="CompleteCertificateRefsType"/>
    <xsd:element name="AttributeRevocationRefs"
      type="CompleteRevocationRefsType"/>
    <xsd:element name="SigAndRefsTimeStamp" type="XAdESTimeStampType"/>
    <xsd:element name="RefsOnlyTimeStamp" type="XAdESTimeStampType"/>
    <xsd:element name="CertificateValues" type="CertificateValuesType"/>
    <xsd:element name="RevocationValues" type="RevocationValuesType"/>
    <xsd:element name="AttrAuthoritiesCertValues"
      type="CertificateValuesType"/>
    <xsd:element name="AttributeRevocationValues"
      type="RevocationValuesType"/>
    <xsd:element name="ArchiveTimeStamp" type="XAdESTimeStampType"/>
    <xsd:any namespace="##other" />
  </xsd:choice>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The optional `Id` attribute can be used to make a reference to the `UnsignedSignatureProperties` element.

The `xsd:any` element MUST be used only for ensuring compatibility among different versions of XAdES. By using this element, applications MAY add any unsigned signature property defined in any other version of XAdES. `It` MUST NOT contain elements whose types and contents are defined outside of of a XAdES specification. This would allow, for instance, the inclusion, of a time-stamp container of one version within an element of another version.

The qualifying property `CounterSignature` is described in detail in clause 7.2.4, `SignatureTimeStamp` in clause 7.3, `CompleteCertificateRefs` in clause 7.4.1, `CompleteRevocationRefs` in clause 7.4.2, `AttributeCertificateRefs` in clause 7.4.3, `AttributeRevocationRefs` in clause 7.4.4, `SigAndRefsTimeStamp` in clause 7.5.1, `RefsOnlyTimeStamp` in clause 7.5.2, `CertificateValues` in clause 7.6.1, `RevocationValues` in clause 7.6.2, `AttrAuthoritiesCertValues` in clause 7.6.3, `AttributeRevocationValues` in clause 7.6.4, and `ArchiveTimeStamp` in clause 7.7.1. These clauses give details on additional restrictions in the number of times that a certain property MAY appear in the different XAdES forms.

### 6.2.6 The `UnsignedDataObjectProperties` element

This element contains properties that qualify some of the signed data objects. The signature generated by the signer does not cover the content of this element.

```
<xsd:element name="UnsignedDataObjectProperties"
  type="UnsignedDataObjectPropertiesType" />

<xsd:complexType name="UnsignedDataObjectPropertiesType">
  <xsd:sequence>
    <xsd:element name="UnsignedDataObjectProperty" type="AnyType"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The optional `Id` attribute can be used to make a reference to the `UnsignedDataObjectProperties` element.

TS 101 733 [1] does not specify the usage of any unsigned property qualifying the signed data object. The present document, however, incorporates this element for the sake of completeness and to cope with potential future needs for inclusion of such kind of properties. The schema definition leaves open the definition of the contents of this type. The type `AnyType` is defined in clause 7.1.1.

## 6.3 Incorporating qualifying properties into an XML signature

The present document utilizes the `ds:Object` auxiliary element from XMLDSIG [3]. It MUST be used to incorporate the qualifying properties into the XMLDSIG signature. In principle, two different means are provided for this incorporation:

- direct incorporation means that a `QualifyingProperties` element is put as a child of the `ds:Object`;
- indirect incorporation means that one or more `QualifyingPropertiesReference` elements appear as children of the `ds:Object`. Each one contains information about one `QualifyingProperties` element that is stored in place different from the signature (see clause 6.3.2).

However, the following restrictions apply for using `ds:Object`, `QualifyingProperties` and `QualifyingPropertiesReference`:

- all instances of the `QualifyingProperties` and the `QualifyingPropertiesReference` elements MUST occur within a single `ds:Object` element;
- at most one instance of the `QualifyingProperties` element MAY occur within this `ds:Object` element;
- all signed properties MUST occur within a single `QualifyingProperties` element. This element can either be a child of this `ds:Object` element (direct incorporation), or it can be referenced by a `QualifyingPropertiesReference` element. See clause 6.3.1 for information how to sign properties;
- zero or more instances of the `QualifyingPropertiesReference` element MAY occur within this `ds:Object` element.

No restrictions apply to the relative position of the `ds:Object` containing the `QualifyingProperties` or `QualifyingPropertiesReference` with respect to others `ds:Object` elements present within `ds:Signature`.

It is out of the scope of the present document to specify the mechanisms required to guarantee the correct storage of the distributed `QualifyingProperties` elements (i.e. that the properties are stored by the entity that has to store them and that they are not undetectable modified).

### 6.3.1 Signing properties

As has already been stated, all the properties that should be protected by the signature have to be collected in a single instance of the `QualifyingProperties` element. Actually these properties are children of the `SignedProperties` child of this element.

In order to protect the properties with the signature, a `ds:Reference` element MUST be added to the XMLDSIG signature. This `ds:Reference` element MUST be composed in such a way that it uses the `SignedProperties` element mentioned above as the input for computing its corresponding digest.

Additionally, the present document MANDATES the use of the `Type` attribute of this particular `ds:Reference` element, with its value set to:

- http://uri.etsi.org/01903#SignedProperties

This value indicates that the data used for hash computation is a `SignedProperties` element and therefore helps a verifying application to detect the signed properties of a signature conforming to the present document.

### 6.3.2 The `QualifyingPropertiesReference` element

This element contains information about a `QualifyingProperties` element that is stored in place different from the signature, for instance in another XML document.

```
<xsd:element name="QualifyingPropertiesReference"
  type="QualifyingPropertiesReferenceType"/>

<xsd:complexType name="QualifyingPropertiesReferenceType">
  <xsd:attribute name="URI" type="xsd:anyURI" use="required"/>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The mandatory `URI` attribute contains a bare-name XPointer fragment and references an external QualifyingProperties element. Its not-fragment part identifies the enclosing document and its bare-name XPointer fragment identifies the aforementioned element.

The optional `Id` attribute can be used to make a reference to the `QualifyingPropertiesReference` element.

# 7 Qualifying properties syntax

This clause describes in detail all qualifying properties which have been introduced in clause 4.3. It provides a rationale for each property as well as its XML Schema definition together with explanatory textual information.

Clause 7.1 summarizes a set of auxiliary structures that will be needed later on, while each of the remaining clauses corresponds to a certain qualifying property.

Clause 7.2 describes in detail the qualifying properties that can appear in XAdES-BES and XAdES-EPES electronic signatures forms as described in clause 4.

Clause 7.3 describes in detail the `SignatureTimeStamp`.

Clause 7.4 describes in detail properties that contain references to validation data.

Clause 7.5 describes in detail properties that can contain time-stamps covering references to validation data.

Clause 7.6 describes in detail properties that can contain validation data values.

Clause 7.7 describes in detail the `ArchivalTimeStamp`.

## 7.1 Auxiliary syntax

Certain auxiliary XML structures, utilized in several cases, are described in the subsequent clauses.

### 7.1.1 The `AnyType` data type

The `AnyType` Schema data type has a content model that allows a sequence of arbitrary XML elements that (mixed with text) is of unrestricted length. It also allows for text content only. Additionally, an element of this data type can bear an unrestricted number of arbitrary attributes. It is used throughout the remaining parts of this clause wherever the content of an XML element has been left open.

```
<xsd:complexType name="AnyType" mixed="true">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:any namespace="##any" processContents="lax"/>
  </xsd:sequence>
  <xsd:anyAttribute namespace="##any"/>
</xsd:complexType>
```

### 7.1.2 The ObjectIdentifierType data type

The ObjectIdentifierType data type can be used to identify a particular data object.

It allows the specification of a unique and permanent identifier of an object. In addition, it may also contain, a textual description of the nature of the data object, and a number of references to documents where additional information about the nature of the data object can be found.

```
<xsd:complexType name="ObjectIdentifierType">
  <xsd:sequence>
    <xsd:element name="Identifier" type="IdentifierType"/>
    <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
    <xsd:element name="DocumentationReferences"
     type="DocumentationReferencesType" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

The Identifier element contains a permanent identifier. Once the identifier is assigned, it can never be re-assigned again. It supports both the mechanism that is used to identify objects in ASN.1 and the mechanism that is usually used to identify objects in an XML environment:

- in a XML environment objects are typically identified by means of a Uniform Resource Identifier, URI. In this case, the content of Identifier consists of the identifying URI, and the optional Qualifier attribute does not appear;
- in ASN.1 an Object IDentifier (OID) is used to identify an object. To support an OID, the content of Identifier consists of an OID, either encoded as Uniform Resource Name (URN) or as Uniform Resource Identifier (URI). The optional Qualifier attribute can be used to provide a hint about the applied encoding (values OIDAsURN or OIDAsURI).

Should an OID and an URI exist identifying the same object, the present document encourages the use of the URI as explained in the first bullet above.

```
<xsd:complexType name="IdentifierType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:anyURI">
      <xsd:attribute name="Qualifier" type="QualifierType"
        use="optional"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:simpleType name="QualifierType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="OIDAsURI"/>
    <xsd:enumeration value="OIDAsURN"/>
  </xsd:restriction>
</xsd:simpleType>
```

The optional Description element contains an informal text describing the object identifier.

The optional DocumentationReferences element consists of an arbitrary number of references pointing to further explanatory documentation of the object identifier.

```
<xsd:complexType name="DocumentationReferencesType">
  <xsd:sequence maxOccurs="unbounded">
    <xsd:element name="DocumentationReference" type="xsd:anyURI"/>
  </xsd:sequence>
</xsd:complexType>
```

### 7.1.3 The EncapsulatedPKIDataType data type

The EncapsulatedPKIDataType is used to incorporate non XML pieces of PKI data into an XML structure. Examples of such PKI data that are widely used at the time being include X.509 certificates and revocation lists, OCSP responses, attribute certificates and time-stamp tokens.

```
<xsd:complexType name="EncapsulatedPKIDataType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:base64Binary">
      <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
      <xsd:attribute name="Encoding" type="xsd:anyURI"
```

```
            use="optional"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

The content of this data type is the piece of PKI data, base64 encoded as defined in [3].

The `Encoding` attribute is an URI identifying the encoding used in the original PKI data. So far, the following URIs have been identified:

http://uri.etsi.org/01903/v1.2.2#DER for denoting that the original PKI data were ASN.1 data encoded in DER.

http://uri.etsi.org/01903/v1.2.2#BER for denoting that the original PKI data were ASN.1 data encoded in BER.

http://uri.etsi.org/01903/v1.2.2#CER for denoting that the original PKI data were ASN.1 data encoded in CER.

http://uri.etsi.org/01903/v1.2.2#PER for denoting that the original PKI data were ASN.1 data encoded in PER.

http://uri.etsi.org/01903/v1.2.2#XER for denoting that the original PKI data were ASN.1 data encoded in XER.

If the `Encoding` attribute is not present, then it is assumed that the PKI data is ASN.1 data encoded in DER.

The optional `ID` attribute can be used to make a reference to an element of this data type.

### 7.1.4 Types for time-stamp tokens management

XAdES uses time-stamp tokens in a number of use cases. The present document defines:

- A XML schema definition of an abstract base type and two concrete derived types used as containers for time-stamp tokens.
- A number of properties of one of the aforementioned concrete types. Time-stamp tokens included in these properties will cover a specific set of elements and properties of XAdES signature forms and will satisfy, in this way, different requirements.

A time-stamp token is obtained by sending the digest value of the given data to the Time-Stamp Authority (TSA). The returned time-stamp token is a signed data that contains the digest value, the identity of the TSA, and the time of stamping. This proves that the given data existed *before* the time of stamping.

XAdES time-stamp tokens container properties contain time-stamp tokens computed on both elements defined in XMLDSIG [3] and properties defined in the present document. The present document uses the term time-stamped data objects for indistinctly denoting any of them.

#### 7.1.4.1 Time-stamp properties in XAdES

Below follows the list of the properties containing time-stamps that are defined by the present document:

- Properties that contain time-stamp tokens proving that some or all the data objects to be signed have been created before some time: `AllDataObjectsTimeStamp` and `IndividualDataObjectsTimeStamp`.
- `SignatureTimeStamp`: it is a container for a time-stamp token over the `SignatureValue` element to protect against repudiation in case of a key compromise.
- Two properties contain time-stamp tokens provided for protection against fraudulence in case of a CA key compromise:
  - `RefsOnlyTimeStamp`: it contains a time-stamp token only over all certificate and revocation information references.
  - `SigAndRefsTimeStamp`: it contains a time-stamp token computed over the signature value, the signature time-stamp and the certificate and revocation information references.
- To provide for long term validity of an XML signature, the signature and validation data values are time-stamped. `ArchiveTimeStamp` is defined for this purpose. More than one instance of this property can be added as time goes on to the archived electronic signature.

### 7.1.4.2 The GenericTimeStampType data type

The abstract base container type for time-stamp tokens specified by the present document does have the following features:

- It may contain encapsulated RFC 3161 [11] time-stamp tokens as well as XML time-stamp tokens.
- It may contain more than one time-stamp token generated for the same XAdES data objects (each one issued by different TSAs, for instance).
- It provides means for managing time-stamp tokens computed on XAdES data objects (as for the aforementioned XAdES properties) or time-stamp tokens computed on external data.
- It may use specific elements for identifying what is time-stamped and how to generate the input data for the computation of the digest value to be sent to the TSA. For certain XAdES data objects under certain circumstances this information may be implicit.

Below follows the schema definition for the data type.

```
<xsd:complexType name="IncludeType">
  <xsd:attribute name="URI" type="xsd:anyURI" use="required"/>
  <xsd:attribute name="referencedData" type="xsd:boolean"
    use="optional"/>
</xsd:complexType>

<xsd:element name="ReferenceInfo" type="ReferenceInfoType"/>
<xsd:complexType name="ReferenceInfoType">
  <xsd:sequence>
    <xsd:element ref="ds:DigestMethod"/>
    <xsd:element ref="ds:DigestValue"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
  <xsd:attribute name="URI" type="xsd:anyURI" use="optional"/>
</xsd:complexType>

<xsd:complexType name="GenericTimeStampType" abstract="true">
  <xsd:sequence>
    <xsd:choice minOccurs="0">
      <xsd:element ref="Include" maxOccurs="unbounded"/>
      <xsd:element ref="ReferenceInfo" maxOccurs="unbounded"/>
    </xsd:choice>
    <xsd:element ref="ds:CanonicalizationMethod" minOccurs="0"/>
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="EncapsulatedTimeStamp"
        type="EncapsulatedPKIDataType"/>
      <xsd:element name="XMLTimeStamp" type="AnyType"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

When present, the optional `ds:CanonicalizationMethod` element will indicate the canonicalization method used for canonicalizing XML node sets resulting after retrieving (and processing when required) the data objects covered by the time-stamp token(s). When not present, the standard canonicalization method as specified by XMLDSIG MUST be used.

The time-stamp token generated by the TSA can be either an ASN.1 data object (as defined in [11], use `EncapsulatedTimeStamp`), or it can be encoded as XML (use `XMLTimeStamp`).

Details on the different elements and supporting types are given in the clauses that define the two concrete types: `XAdESTimeStampType` and `OtherTimeStampType`.

### 7.1.4.3 The XAdESTimeStampType data type

This concrete derived type is provided for containing time-stamp tokens computed on data objects of XAdES signatures. Applications claiming alignment with the present document MUST implement it because all the properties listed in clause 7.1.4.1 are elements of this type.

Below follows the schema definition for the data type.

```
<xsd:element name="XAdESTimeStamp" type="XAdESTimeStampType"/>

<xsd:complexType name="XAdESTimeStampType">
  <xsd:complexContent>
    <xsd:restriction base="GenericTimeStampType">
      <xsd:sequence>
        <xsd:element ref="Include" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element ref="ds:CanonicalizationMethod" minOccurs="0"/>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="EncapsulatedTimeStamp"
            type="EncapsulatedPKIDataType"/>
          <xsd:element name="XMLTimeStamp" type="AnyType"/>
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

This type provides two mechanisms for identifying data objects that are covered by the time-stamp token present in the container, and for specifying how to use them for computing the digest value that is sent to the TSA:

- Explicit. This mechanism uses the `Include` element for referencing specific data objects and for indicating their contribution to the input of the digest computation.
- Implicit. For certain time-stamp container properties under certain circumstances, applications do not require any additional indication for knowing that certain data objects are covered by the time-stamp tokens and how they contribute to the input of the digest computation. The present document specifies, in the clauses defining such properties (clauses 7.2.9, 7.2.10, 7.3, 7.5 and 7.7), how applications MUST act in these cases without explicit indications.

Clause 7.1.4.3.1 shows the principles that govern the explicit indication mechanism.

#### 7.1.4.3.1 Include mechanism

`Include` elements explicitly identify data objects that are time-stamped. Their order of appearance indicates how the data objects contribute in the generation of the input to the digest computation. The following time-stamp token container properties use this mechanism:

- `IndividualDataObjectsTimeStamp`. In this case each `Include` element contains an URI to one of the `ds:Reference` elements in the XAdES signature.
- `SigAndRefsTimeStamp, RefsOnlyTimeStamp` and `ArchiveTimeStamp` only and only if these elements and some of the unsigned properties covered by their time-stamp tokens do not have the same parent (i.e. some of them are in different `QualifyingProperties` elements and the XAdES signature contains `QualifyingPropertiesReference` elements - see clause 6.3).

The `URI` attribute in `Include` element identifies one time-stamped data object. Its value MUST follow the rules indicated below:

- It MUST have an empty not-fragment part and a bare-name XPointer fragment when the `Include` and the time-stamped data object are in the same document.
- It MUST have a not empty not-fragment part and a bare-name XPointer fragment when the `Include` and the time-stamped data object are not in the same document.
- When not empty, its not-fragment part MUST be equal to:
  - The not-fragment part of the `Target` attribute of the `QualifyingProperties` enclosing the `Include` element if the time-stamped data object is enveloped by the XAdES signature, or
  - The not-fragment part of the `URI` attribute of the `QualifyingPropertiesReference` element referencing the `QualifyingProperties` element enveloping the time-stamped data object if this `QualifyingProperties` element is not enveloped by the XAdES signature.

Applications aligned with the present specification MUST parse the retrieved resource, and then process the bare-name XPointer as explained below to get a XPath node-set suitable for use by Canonical XML. For processing the bare-name XPointer applications MUST use as XPointer evaluation context the root node of the XML document that contains the element referenced by the not-fragment part of URI. Applications MUST derive an XPath node-set from the resultant location-set as indicated below:

1) Replace the element node E retrieved by the bare-name XPointer with E plus all descendants of E (text, comments, PIs, elements) and all namespace and attribute nodes of E and its descendant elements.

2) Delete all the comment nodes.

In time-stamps that cover `ds:Reference` elements, the attribute `referencedData` MAY be present. If present with value set to `"true"`, the time-stamp is computed on the result of processing the corresponding `ds:Reference` element according to the XMLDSIG processing model. If the attribute is not present or is present with value `"false"`, the time-stamp is computed on the `ds:Reference` element itself. When appearing in a time-stamp container property, each `Include` element MUST be processed in order as detailed below:

1) Retrieve the data object referenced in the `URI` attribute following the referencing mechanism indicated above.

2) If the retrieved data is a `ds:Reference` element and the `referencedData` attribute is set to the value `"true"`, take the result of processing the retrieved `ds:Reference` element according to the reference processing model of XMLDSIG; otherwise take the `ds:Reference` element itself.

3) If the resulting data is an XML node set, canonicalize it. If `ds:Canonicalization` is present, the algorithm indicated by this element is used. If not, the standard canonicalization method specified by XMLDSIG is used.

4) Concatenate the resulting octets to those resulting from previous processing as indicated in the corresponding time-stamp container property.

#### 7.1.4.4 The OtherTimeStampType data type

This concrete derived type is provided for containing time-stamp tokens computed on a collection of data objects that are not present in the XAdES signature.

Below follows the schema definition for the data type.

```
<xsd:element name="OtherTimeStamp" type="OtherTimeStampType"/>

<xsd:complexType name="OtherTimeStampType">
    <xsd:complexContent>
        <xsd:restriction base="GenericTimeStampType">
            <xsd:sequence>
                <xsd:element ref="ReferenceInfo"  maxOccurs="unbounded"/>
                <xsd:element ref="ds:CanonicalizationMethod" minOccurs="0"/>
                <xsd:choice>
                    <xsd:element name="EncapsulatedTimeStamp"
              type="EncapsulatedPKIDataType"/>
                    <xsd:element name="XMLTimeStamp" type="AnyType"/>
            </xsd:choice>
            </xsd:sequence>
            <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Each `ReferenceInfo` element contains the digest of one external data object. Attribute `URI` identifies the data object. As in XMLDSIG, if it is omitted, the application is expected to know the identity of the referenced object. Attribute `Id` permits this element to be referenced from elsewhere. Element `ds:DigestMethod` identifies the digest algorithm applied to the external data object. Element `ds:DigestValue` contains the base64 encoded value of the digest of the referenced data object.

Attribute `Id` and elements `ds:CannonicalizationMethod`, `EncapsulatedTimeStamp` and `XMLTimeStamp` will be used exactly as in `XAdESTimeStampType`.

For this type the actual input to the computation of the digest value that will be sent to the TSA is the concatenation of the canonicalized present `ReferenceInfo` elements. If `ds:Canonicalization` is present, the algorithm indicated by this element is used. If not, the standard canonicalization method specified by XMLDSIG is used.

The implementation of such a type is NOT MANDATORY for applications that claim alignment with the present document, as it does not define any property of this type.

## 7.2 Properties for XAdES-BES and XAdES-EPES forms

This clause describes in detail the qualifying properties that can appear in XAdES-BES and XAdES-EPES forms as described in clauses 4.4.1 and 4.4.2.

### 7.2.1 The `SigningTime` element

The `SigningTime` property specifies the time at which the signer (purportedly) performed the signing process.

The XML Schema recommendation [5] defines an XML type `xsd:dateTime` that allows for the inclusion of the required information. This is the type selected for the `SigningTime` element.

This is a signed property that qualifies the whole signature.

At most one `SigningTime` element MAY be present in the signature.

Below follows the Schema definition for this element.

```
<xsd:element name="SigningTime" type="xsd:dateTime"/>
```

### 7.2.2 The `SigningCertificate` element

In many real life environments users will be able to get from different CAs or even from the same CA, different certificates containing the same public key for different names. The prime advantage is that a user can use the same private key for different purposes. Multiple use of the private key is an advantage when a smart card is used to protect the private key, since the storage of a smart card is always limited. When several CAs are involved, each different certificate may contain a different identity, e.g. as a national or as an employee from a company. Thus when a private key is used for various purposes, the certificate is needed to clarify the context in which the private key was used when generating the signature. Where there is the possibility of multiple uses of private keys it is necessary for the signer to indicate to the verifier the precise certificate to be used.

Many current schemes simply add the certificate after the signed data and thus are subject to various substitution attacks. An example of a substitution attack is a "bad" CA that would issue a certificate to someone with the public key of someone else. If the certificate from the signer was simply appended to the signature and thus not protected by the signature, any one could substitute one certificate by another and the message would appear to be signed by some one else. In order to counter this kind of attack, the identifier of the certificate has to be protected by the digital signature from the signer.

The `SigningCertificate` property is designed to prevent the simple substitution of the certificate. This property contains references to certificates and digest values computed on them.

The certificate used to verify the signature SHALL be identified in the sequence; the signature policy MAY mandate other certificates be present, that MAY include all the certificates up to the point of trust.

This is a signed property that qualifies the signature.

At most one `SigningCertificate` element MAY be present in the signature.

Below follows the Schema definition.

```
<xsd:element name="SigningCertificate" type="CertIDListType"/>

<xsd:complexType name="CertIDListType">
    <xsd:sequence>
        <xsd:element name="Cert" type="CertIDType"
       maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="CertIDType">
    <xsd:sequence>
        <xsd:element name="CertDigest" type="DigestAlgAndValueType"/>
        <xsd:element name="IssuerSerial" type="ds:X509IssuerSerialType"/>
    </xsd:sequence>
    <xsd:attribute name="URI" type="xsd:anyURI" use="optional"/>
</xsd:complexType>

<xsd:complexType name="DigestAlgAndValueType">
    <xsd:sequence>
        <xsd:element ref="ds:DigestMethod"/>
        <xsd:element ref="ds:DigestValue"/>
    </xsd:sequence>
</xsd:complexType>
```

The `SigningCertificate` element contains the aforementioned sequence of certificate identifiers and digests computed on the certificates (`Cert` elements).

The element `IssuerSerial` contains the identifier of one of the certificates referenced in the sequence. Should the `ds:X509IssuerSerial` element appear in the signature to denote the same certificate, its value MUST be consistent with the corresponding `IssuerSerial` element.

The element `CertDigest` contains the digest of one of the certificates referenced in the sequence. It contains two elements: `ds:DigestMethod` indicates the digest algorithm and `ds:DigestValue` contains the base64 encoded value of the digest.

The optional `URI` attribute indicates where the referenced certificate can be found.

### 7.2.3 The `SignaturePolicyIdentifier` element

The signature policy is a set of rules for the creation and validation of an electronic signature, under which the signature can be determined to be valid. A given legal/contractual context MAY recognize a particular signature policy as meeting its requirements.

The signature policy needs to be available in human readable form so that it can be assessed to meet the requirements of the legal and contractual context in which it is being applied.

To facilitate the automatic processing of an electronic signature the parts of the signature policy which specify the electronic rules for the creation and validation of the electronic signature also need to be in a computer processable form.

If no signature policy is identified then the signature may be assumed to have been generated/verified without any policy constraints, and hence may be given no specific legal or contractual significance through the context of a signature policy.

The present document specifies two unambiguous ways for identifying the signature policy that a signature follows:

- The electronic signature can contain an explicit and unambiguous identifier of a signature policy together with a hash value of the signature policy, so it can be verified that the policy selected by the signer is the one being used by the verifier. An explicit signature policy has a globally unique reference, which, in this way, is bound to an electronic signature by the signer as part of the signature calculation. In these cases, for a given explicit signature policy there shall be one definitive form that has a unique binary encoded value. Finally, a signature policy identified in this way MAY be qualified by additional information.

- Alternatively, the electronic signature can avoid the inclusion of the aforementioned identifier and hash value. This will be possible when the signature policy can be unambiguously derived from the semantics of the type of data object(s) being signed, and some other information, e.g. national laws or private contractual agreements, that mention that a given signature policy MUST be used for this type of data content. In such cases, the signature will contain a specific empty element indicating that this implied way to identify the signature policy is used instead the identifier and hash value.

The signature policy identifier is a signed property qualifying the signature.

At most one `SignaturePolicyIdentifier` element MAY be present in the signature.

Below follows the Schema definition for this type.

```
<xsd:element name="SignaturePolicyIdentifier" type="SignaturePolicyIdentifierType"/>

<xsd:complexType name="SignaturePolicyIdentifierType">
  <xsd:choice>
    <xsd:element name="SignaturePolicyId" type="SignaturePolicyIdType"/>
    <xsd:element name="SignaturePolicyImplied"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="SignaturePolicyIdType">
  <xsd:sequence>
    <xsd:element name="SigPolicyId" type="ObjectIdentifierType"/>
    <xsd:element ref="ds:Transforms" minOccurs="0"/>
    <xsd:element name="SigPolicyHash" type="DigestAlgAndValueType"/>
    <xsd:element name="SigPolicyQualifiers"
      type="SigPolicyQualifiersListType" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SigPolicyQualifiersListType">
  <xsd:sequence>
    <xsd:element name="SigPolicyQualifier" type="AnyType"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

The `SignaturePolicyId` element will appear when the signature policy is identified using the first alternative. The `SigPolicyId` element contains an identifier that uniquely identifies a specific version of the signature policy. The `SigPolicyHash` element contains the identifier of the hash algorithm and the hash value of the signature policy. The `SigPolicyQualifier` element can contain additional information qualifying the signature policy identifier. The optional `ds:Transforms` element can contain the transformations performed on the signature policy document before computing its hash. The processing model for these transformations is described in [3].

Alternatively, the `SignaturePolicyImplied` element will appear when the second alternative is used. This empty element indicates that the data object(s) being signed and other external data imply the signature policy.

### 7.2.3.1 Signature Policy qualifiers

Two qualifiers for the signature policy have been identified so far:

- a URL where a copy of the signature policy MAY be obtained;
- a user notice that should be displayed when the signature is verified.

Below follows the Schema definition for these two elements.

```
<xsd:element name="SPURI" type="xsd:anyURI"/>
<xsd:element name="SPUserNotice" type="SPUserNoticeType"/>

<xsd:complexType name="SPUserNoticeType">
  <xsd:sequence>
    <xsd:element name="NoticeRef" type="NoticeReferenceType"
      minOccurs="0"/>
    <xsd:element name="ExplicitText" type="xsd:string"
      minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="NoticeReferenceType">
  <xsd:sequence>
    <xsd:element name="Organization" type="xsd:string"/>
    <xsd:element name="NoticeNumbers" type="IntegerListType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="IntegerListType">
  <xsd:sequence>
    <xsd:element name="int" type="xsd:integer" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

The SPUserNotice element is intended for being displayed whenever the signature is validated. The ExplicitText element contains the text of the notice to be displayed. Other notices could come from the organization issuing the signature policy. The NoticeRef element names an organization and identifies by numbers (NoticeNumbers element) a group of textual statements prepared by that organization, so that the application could get the explicit notices from a notices file.

### 7.2.4 Countersignatures

Annex C, in its clause C.1, includes a description and discussion on multiple signatures and countersignatures. This clause defines two standard mechanisms for managing countersignatures. Details are given in clauses below.

#### 7.2.4.1 Countersignature identifier in Type attribute of ds:Reference

The present document defines the following URI value:

- http://uri.etsi.org/01903#CountersignedSignature

A XAdES signature containing a ds:Reference element whose Type attribute has this value will indicate that it is, in fact, a countersignature of the signature referenced by this element. The ds:Reference element MUST be built so that the countersignature actually signs the ds:SignatureValue element of the countersigned signature. All the XMLDSIG rules apply in the processing of the aforementioned ds:Reference element. The only purpose of this definition is to serve as an easy identification of a signature as actually being a countersignature.

#### 7.2.4.2 Enveloped countersignatures: the CounterSignature element

The CounterSignature is an unsigned property that qualifies the signature. A XAdES signature MAY have more than one CounterSignature properties. As indicated by its name, it contains one countersignature of the qualified signature.

The content of this property is a XMLDSIG or XAdES signature whose ds:SignedInfo MUST contain one ds:Reference element referencing the ds:SignatureValue element of the embedding and countersigned XAdES signature. The content of the ds:DigestValue in the aforementioned ds:Reference element of the countersignature MUST be the base64 encoded digest of the complete (and canonicalized) ds:SignatureValue element (i.e. including the starting and closing tags) of the embedding and countersigned XAdES signature. Applications MUST build this ds:Reference accordingly, using any of the mechanisms specified by XMLDSIG for achieving this objective. By doing this the countersignature actually signs the ds:SignatureValue element of the embedding XAdES signature.

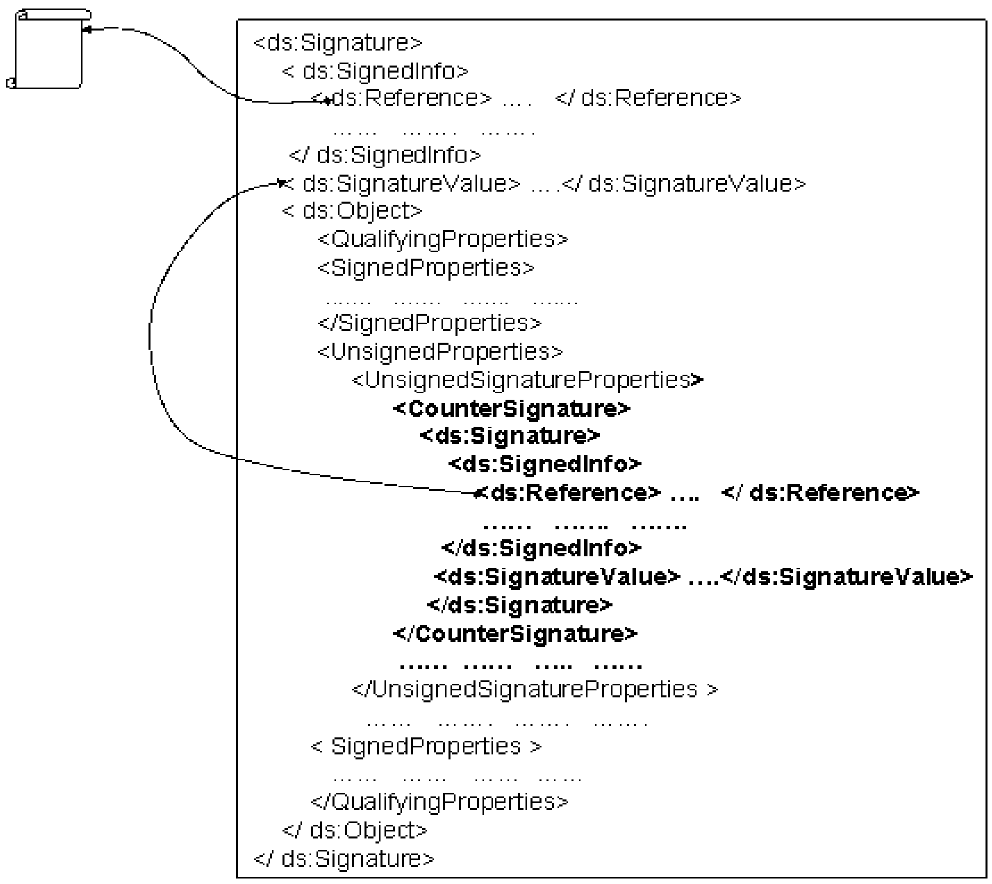
Applications MAY add other ds:Reference elements referencing the ds:SignatureValue elements of previously existent CounterSignature elements. This allows for building arbitrarily long chains of explicit countersignatures.

A countersignature MAY itself be qualified by a CounterSignature property, which will have a ds:Reference element referencing the ds:SignatureValue of the first countersignature, built as described above. This is an alternative way of constructing arbitrarily long series of countersignatures, each one signing the ds:SignatureValue element of the one where it is directly embedded.

If the countersignature is a XAdES signature, its production MUST follow the rules dictated by the current specification. As for its verification they MUST be verified as any regular XAdES signature. Below follows the schema definition for this element.

```
<xsd:element name="CounterSignature" type="CounterSignatureType" />

<xsd:complexType name="CounterSignatureType">
    <xsd:sequence>
        <xsd:element ref="ds:Signature"/>
    </xsd:sequence>
</xsd:complexType>
```



```
<ds:Signature>
  < ds:SignedInfo>
    <ds:Reference> ....  </ ds:Reference>
      ......  ........  ........
   </ ds:SignedInfo>
  < ds:SignatureValue> ....</ ds:SignatureValue>
  < ds:Object>
     <QualifyingProperties>
     <SignedProperties>
      .......  .......  .......  .......
     </SignedProperties>
     <UnsignedProperties>
        <UnsignedSignatureProperties>
           <CounterSignature>
             <ds:Signature>
               <ds:SignedInfo>
                 <ds:Reference> ....   </ ds:Reference>
                   ......  .......  .......
               </ds:SignedInfo>
              <ds:SignatureValue> ....</ds:SignatureValue>
             </ds:Signature>
           </CounterSignature>
           ......  ......  .....  ......
        </UnsignedSignatureProperties >
          ......  ........  ........  ........
     < SignedProperties >
       ......  ......  ......  ......
     </QualifyingProperties>
  </ ds:Object>
</ ds:Signature>
```

**Figure 2: Use of CounterSignature element**

## 7.2.5 The `DataObjectFormat` element

When presenting signed data to a human user it may be important that there is no ambiguity as to the presentation of the signed data object to the relying party. In order for the appropriate representation (text, sound or video) to be selected by the relying party a content hint MAY be indicated by the signer. If a relying party system does not use the format specified to present the data object to the relying party, the electronic signature may not be valid. Such behaviour may have been established by the signature policy, for instance.

The `DataObjectFormat` element provides information that describes the format of the signed data object. This element SHOULD be present when the signed data is to be presented to human users on verification if the presentation format is not implicit within the data that has been signed. This is a signed property that qualifies one specific signed data object. In consequence, an XML electronic signature aligned with the present document MAY contain more than one `DataObjectFormat` elements, each one qualifying one signed data object.

Below follows the schema definition for this element.

```
<xsd:element name="DataObjectFormat" type="DataObjectFormatType"/>

<xsd:complexType name="DataObjectFormatType">
    <xsd:sequence>
        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
        <xsd:element name="ObjectIdentifier" type="ObjectIdentifierType"
      minOccurs="0"/>
        <xsd:element name="MimeType" type="xsd:string" minOccurs="0"/>
        <xsd:element name="Encoding" type="xsd:anyURI" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="ObjectReference" type="xsd:anyURI"
   use="required"/>
</xsd:complexType>
```

The mandatory `ObjectReference` attribute MUST reference the `ds:Reference` element of the `ds:Signature` corresponding with the data object qualified by this property.

This element can convey:

- textual information related to the signed data object in element `Description`;
- an identifier indicating the type of the signed data object in element `ObjectIdentifier`;
- an indication of the MIME type of the signed data object in element `MimeType`;
- an indication of the encoding format of the signed data object in element `Encoding`.

At least one element of `Description`, `ObjectIdentifier` and `MimeType` MUST be present within the property.

### 7.2.6 The `CommitmentTypeIndication` element

The commitment type can be indicated in the electronic signature either:

- explicitly using a commitment type indication in the electronic signature;
- implicitly or explicitly from the semantics of the signed data object.

If the indicated commitment type is explicit by means of a commitment type indication in the electronic signature, acceptance of a verified signature implies acceptance of the semantics of that commitment type. The semantics of explicit commitment types indications shall be specified either as part of the signature policy or MAY be registered for generic use across multiple policies.

If a signature includes a commitment type indication other than one of those recognized under the signature policy the signature shall be treated as invalid.

How commitment is indicated using the semantics of the data object being signed is outside the scope of the present document.

The commitment type MAY be:

- defined as part of the signature policy, in which case the commitment type has precise semantics that is defined as part of the signature policy;
- a registered type, in which case the commitment type has precise semantics defined by registration, under the rules of the registration authority. Such a registration authority may be a trading association or a legislative authority.

The definition of a commitment type includes:

- the object identifier for the commitment;
- a sequence of qualifiers.

The qualifiers can provide more information about the commitment, it could provide, for example, information about the context be it contractual/legal/application specific.

If an electronic signature does not contain a recognized commitment type then the semantics of the electronic signature is dependent on the data object being signed and the context in which it is being used.

This is a signed property that qualifies signed data object(s). In consequence, an XML electronic signature aligned with the present document MAY contain more than one `CommitmentTypeIndication` elements.

Below follows the schema definition for this element.

```
<xsd:element name="CommitmentTypeIndication" type="CommitmentTypeIndicationType"/>

<xsd:complexType name="CommitmentTypeIndicationType">
    <xsd:sequence>
        <xsd:element name="CommitmentTypeId"
       type="ObjectIdentifierType"/>
        <xsd:choice>
            <xsd:element name="ObjectReference" type="xsd:anyURI"
         maxOccurs="unbounded"/>
```

```
            <   xsd:element name="AllSignedDataObjects"/>
        </xsd:choice>
        <xsd:element name="CommitmentTypeQualifiers"
      type="CommitmentTypeQualifiersListType" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CommitmentTypeQualifiersListType">
    <xsd:sequence>
        <xsd:element name="CommitmentTypeQualifier"
      type="AnyType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

The `CommitmentTypeId` element univocally identifies the type of commitment made by the signer. A number of commitments have been already identified in TS 101 733 [1], namely:

- **Proof of origin** indicates that the signer recognizes to have created, approved and sent the signed data object. The URI for this commitment is http://uri.etsi.org/01903/v1.2.2#ProofOfOrigin.
- **Proof of receipt** indicates that signer recognizes to have received the content of the signed data object. The URI for this commitment is http://uri.etsi.org/01903/v1.2.2#ProofOfReceipt.
- **Proof of delivery** indicates that the TSP providing that indication has delivered a signed data object in a local store accessible to the recipient of the signed data object. The URI for this commitment is http://uri.etsi.org/01903/v1.2.2#ProofOfDelivery.
- **Proof of sender** indicates that the entity providing that indication has sent the signed data object (but not necessarily created it). The URI for this commitment is http://uri.etsi.org/01903/v1.2.2#ProofOfSender.
- **Proof of approval** indicates that the signer has approved the content of the signed data object. The URI for this commitment is http://uri.etsi.org/01903/v1.2.2#ProofOfApproval.
- **Proof of creation** indicates that the signer has created the signed data object (but not necessarily approved, nor sent it). The URI for this commitment is http://uri.etsi.org/01903/v1.2.2#ProofOfCreation.

One `ObjectReference` element refers to one `ds:Reference` element of the `ds:SignedInfo` corresponding with one data object qualified by this property. If some but not all the signed data objects share the same commitment, one `ObjectReference` element MUST appear for each one of them. However, if all the signed data objects share the same commitment, the `AllSignedDataObjects` empty element MUST be present.

The `CommitmentTypeQualifiers` element provides means to include additional qualifying information on the commitment made by the signer.

### 7.2.7 The `SignatureProductionPlace` element

In some transactions the purported place where the signer was at the time of signature creation MAY need to be indicated. In order to provide this information a new property MAY be included in the signature.

This property specifies an address associated with the signer at a particular geographical (e.g. city) location.

This is a signed property that qualifies the signer.

An XML electronic signature aligned with the present document MAY contain at most one `SignatureProductionPlace` element.

Below follows the schema definition for this element.

```
<xsd:element name="SignatureProductionPlace" type="SignatureProductionPlaceType"/>

<xsd:complexType name="SignatureProductionPlaceType">
    <xsd:sequence>
        <xsd:element name="City" type="xsd:string" minOccurs="0"/>
        <xsd:element name="StateOrProvince" type="xsd:string"
       minOccurs="0"/>
        <xsd:element name="PostalCode" type="xsd:string" minOccurs="0"/>
        <xsd:element name="CountryName" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

## 7.2.8 The `SignerRole` element

While the name of the signer is important, the position of the signer within a company or an organization can be even more important. Some contracts may only be valid if signed by a user in a particular role, e.g. a Sales Director. In many cases who the sales Director really is, is not that important but being sure that the signer is empowered by his company to be the Sales Director is fundamental.

The present document defines two different ways for providing this feature:

- using a claimed role name;
- using an attribute certificate containing a *certified* role.

The signer MAY state his own role without any certificate to corroborate this claim, in which case the claimed role can be added to the signature as a signed qualifying property.

Unlike public key certificates that bind an identifier to a public key, Attribute Certificates bind the identifier of a certificate to some attributes of its owner, like a role. The Attribute Authority will be most of the time under the control of an organization or a company that is best placed to know which attributes are relevant for which individual. The Attribute Authority MAY use or point to public key certificates issued by any CA, provided that the appropriate trust may be placed in that CA. Attribute Certificates MAY have various periods of validity. That period may be quite short, e.g. one day. While this requires that a new Attribute Certificate is obtained every day, valid for that day, this can be advantageous since revocation of such certificates may not be needed. When signing, the signer will have to specify which Attribute Certificate it selects.

This is a signed property that qualifies the signer.

An XML electronic signature aligned with the present document MAY contain at most one `SignerRole` element.

Below follows the Schema definition for this element.

```
<xsd:element name="SignerRole" type="SignerRoleType"/>

<xsd:complexType name="SignerRoleType">
  <xsd:sequence>
    <xsd:element name="ClaimedRoles" type="ClaimedRolesListType"
      minOccurs="0"/>
    <xsd:element name="CertifiedRoles" type="CertifiedRolesListType"
      minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ClaimedRolesListType">
  <xsd:sequence>
    <xsd:element name="ClaimedRole" type="AnyType"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CertifiedRolesListType">
  <xsd:sequence>
    <xsd:element name="CertifiedRole" type="EncapsulatedPKIDataType"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

This property contains a sequence of roles that the signer can play (element `SignerRole`). At least one of the two elements `ClaimedRoles` or `CertifiedRoles` MUST be present.

The `ClaimedRoles` element contains a sequence of roles claimed by the signer but not certified. Additional contents types MAY be defined on a domain application basis and be part of this element. The namespaces given to the corresponding XML schemas will allow their unambiguous identification in the case these roles use XML.

The `CertifiedRoles` element contains one or more wrapped DER-encoded attribute certificates for the signer.

### 7.2.9 The AllDataObjectsTimeStamp element

The AllDataObjectsTimeStamp element contains the time-stamp computed before the signature production, over the sequence formed by ALL the ds:Reference elements within the ds:SignedInfo referencing whatever the signer wants to sign except the SignedProperties element.

The AllDataObjectsTimeStamp element is a signed property. Several instances of this property from different TSAs can occur within the same XAdES.

Below follows the schema definition for this element.

```
<xsd:element name="AllDataObjectsTimeStamp" type="XAdESTimeStampType"/>
```

This property uses the Implicit mechanism. The input to the computation of the digest value MUST be the result of processing the aforementioned suitable ds:Reference elements in their order of appearance within ds:SignedInfo as follows:

1) Process the retrieved ds:Reference element according to the reference processing model of XMLDSIG.
2) If the result is a XML node set, canonicalize it. If ds:Canonicalization is present, the algorithm indicated by this element is used. If not, the standard canonicalization method specified by XMLDSIG is used.
3) Concatenate the resulting octets to those resulting from previously processed ds:Reference elements in ds:SignedInfo.

### 7.2.10 The IndividualDataObjectsTimeStamp element

The IndividualDataObjectsTimeStamp element contains the time-stamp computed before the signature production, over a sequence formed by SOME ds:Reference elements within the ds:SignedInfo. Note that this sequence cannot contain a ds:Reference computed on the SignedProperties element.

The IndividualDataObjectsTimeStamp element is a signed property that qualifies the signed data object(s).

Several instances of this property can occur within the same XAdES.

Below follows the schema definition for this element.

```
<xsd:element name="IndividualDataObjectsTimeStamp"
  type="XAdESTimeStampType"/>
```

This property uses the explicit (Include) mechanism. Generating applications MUST compose the Include elements to refer to those ds:Reference elements that are to be time-stamped. Their corresponding referencedData attribute MUST be present and set to "true".

The digest computation input MUST be the result of processing the selected ds:Reference within ds:SignedInfo as follows:

1) Process the retrieved ds:Reference element according to the reference processing model of XMLDSIG.
2) If the result is a XML node set, canonicalize it. If ds:Canonicalization is present, the algorithm indicated by this element is used. If not, the standard canonicalization method specified by XMLDSIG is used.
3) Concatenate the resulting octets to those resulting from previously processed ds:Reference elements in ds:SignedInfo.

## 7.3 The SignatureTimeStamp element

An important property for long standing signatures is that a signature, having been found once to be valid, shall continue to be so months or years later.

A signer, verifier or both MAY be required to provide on request, proof that a digital signature was created or verified during the validity period of all the certificates that make up the certificate path. In this case, the signer, verifier or both will also be required to provide proof that all the user and CA certificates used were not revoked when the signature was created or verified.

It would be quite unacceptable to consider a signature as invalid even if the keys or certificates were only compromised later. Thus there is a need to be able to demonstrate that the signature key was valid around the time that the signature was created to provide long term evidence of the validity of a signature. Time-stamping by a Time-Stamping Authority (TSA) can provide such evidence.

Time-stamping an electronic signature before the revocation of the signer's private key and before the end of the validity of the certificate provides evidence that the signature has been created while the certificate was valid and before it was revoked.

If a recipient wants to keep the result of the validation of an electronic signature valid, he will have to ensure that he has obtained a valid time-stamp for it, before that key (and any key involved in the validation) is revoked. The sooner the time-stamp is obtained after the signing time, the better.

It is important to note that signatures MAY be generated "off-line" and time-stamped at a later time by anyone, for example by the signer or any recipient interested in the signature. The time-stamp can thus be provided by the signer together with the signed data object, or obtained by the recipient following receipt of the signed data object.

The validation mandated by the signature policy can specify a maximum acceptable time difference which is allowed between the time indicated in the `SigningTime` element and the time indicated by the `SignatureTimeStamp` element. If this delay is exceeded then the electronic signature shall be considered as invalid.

The `SignatureTimeStamp` encapsulates the time-stamp over the `ds:SignatureValue` element.

This property uses the implicit mechanism as the time-stamped data object is always the same. For building the input to the digest computation, applications MUST:

1) Take the `ds:SignatureValue` element and its contents.

2) If the `ds:Canonicalization` element is present canonicalize it using the indicated algorithm. If not, use the standard canonicalization method specified by XMLDSIG.

The `SignatureTimeStamp` element is an unsigned property qualifying the signature. A XAdES-T form MAY contain several `SignatureTimeSamp` elements, obtained from different TSAs.

Below follows the schema definition for this element.

```
<xsd:element name="SignatureTimeStamp" type="XAdESTimeStampType"/>
```

## 7.4 Properties for references to validation data

The following clauses describe in detail qualifying properties that can contain references to certificates and revocation values that have been used in the validation of the electronic signature.

When dealing with long term electronic signatures, all the data used in the verification (namely, certificate path and revocation information) of such signatures MUST be stored and conveniently time-stamped for arbitration purposes. Similar considerations apply to attribute certificates if they appear within the signature.

In some environments, it can be convenient to add these data to the electronic signature (as unsigned properties) for archival purposes (see electronic signature form for archival in clause B.3).

Systems implementing the present document may consider to archive validation data outside the XAdES e.g. to prevent redundant storage and to reduce the size of the signatures. In such cases each electronic signature MUST incorporate references to all these data within the signature, reducing accordingly the size of the stored electronic signature. This format builds up taking XAdES-T signature by incorporating additional data required for validation:

- the sequence of references to the full set of CA certificates that have been used to validate the electronic signature up to (but not including) the signer's certificate;
- the sequence of references to the full set of revocation data that have been used in the validation of the signer and CA certificates;

- the references to the full set of Attribute Authorities certificates that have been used to validate the attribute certificate, if present;
- the references to the full set of revocation data that have been used in the validation of the attribute certificate, if present.

### 7.4.1 The `CompleteCertificateRefs` element

This clause defines the XML element able to carry the aforementioned references to the CA certificates: the `CompleteCertificateRefs` element.

This is an unsigned property that qualifies the signature.

An XML electronic signature aligned with the present document MAY contain at most one `CompleteCertificateRefs` element.

Below follows the schema definition for this element.

```
<xsd:element name="CompleteCertificateRefs" type="CompleteCertificateRefsType"/>

<xsd:complexType name="CompleteCertificateRefsType">
  <xsd:sequence>
    <xsd:element name="CertRefs" type="CertIDListType" />
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The `CertRefs` element contains a sequence of `Cert` elements already defined in clause 7.2.2, incorporating the digest of each certificate and the issuer and serial number identifier.

Should XML time-stamp tokens based in XMLDSIG be standardized and spread, this type could also serve to contain references to the certification chain for any TSUs providing such time-stamp tokens. In this case, an element of this type could be added as an unsigned property to the XML time-stamp token using the incorporation mechanisms defined in the present specification.

### 7.4.2 The `CompleteRevocationRefs` element

As it was stated in the previous clause, the addition, to an electronic signature, of the full set of references to the revocation data that have been used in the validation of the signer and CAs certificates, provide means to retrieve the actual revocation data archived elsewhere in case of dispute and, in this way, to illustrate that the verifier has taken due diligence of the available revocation information.

Currently two major types of revocation data are managed in most of the systems, namely CRLs and responses of on-line certificate status servers, obtained through protocols designed for these purposes, like OCSP protocol.

This clause defines the `CompleteRevocationRefs` element that will carry the full set of revocation information used for the verification of the electronic signature.

This is an unsigned property that qualifies the signature.

An XML electronic signature aligned with the present document MAY contain at most one `CompleteRevocationRefs` element.

Below follows the Schema definition for this element.

```
<xsd:element name="CompleteRevocationRefs"
  type="CompleteRevocationRefsType"/>

<xsd:complexType name="CompleteRevocationRefsType">
  <xsd:sequence>
    <xsd:element name="CRLRefs" type="CRLRefsType" minOccurs="0"/>
    <xsd:element name="OCSPRefs" type="OCSPRefsType" minOccurs="0"/>
    <xsd:element name="OtherRefs" type="OtherCertStatusRefsType"
      minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

```
<xsd:complexType name="CRLRefsType">
  <xsd:sequence>
    <xsd:element name="CRLRef" type="CRLRefType"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CRLRefType">
  <xsd:sequence>
    <xsd:element name="DigestAlgAndValue"
      type="DigestAlgAndValueType"/>
    <xsd:element name="CRLIdentifier" type="CRLIdentifierType"
      minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CRLIdentifierType">
  <xsd:sequence>
    <xsd:element name="Issuer" type="xsd:string"/>
    <xsd:element name="IssueTime" type="xsd:dateTime" />
    <xsd:element name="Number" type="xsd:integer"  minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="URI" type="xsd:anyURI" use="optional"/>
</xsd:complexType>

<xsd:complexType name="OCSPRefsType">
  <xsd:sequence>
    <xsd:element name="OCSPRef" type="OCSPRefType"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="OCSPRefType">
  <xsd:sequence>
    <xsd:element name="OCSPIdentifier" type="OCSPIdentifierType"/>
    <xsd:element name="DigestAlgAndValue"
      type="DigestAlgAndValueType"
      minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

    <xsd:complexType name="ResponderIDType">
        <xsd:choice>
            <xsd:element name="ByName" type="xsd:string"/>
            <xsd:element name="ByKey" type="xsd:base64Binary"/>
        </xsd:choice>
    </xsd:complexType>

<xsd:complexType name="OCSPIdentifierType">
  <xsd:sequence>
    <xsd:element name="ResponderID" type="ResponderIDType"/>
    <xsd:element name="ProducedAt" type="xsd:dateTime"/>
  </xsd:sequence>
  <xsd:attribute name="URI" type="xsd:anyURI" use="optional"/>
</xsd:complexType>

<xsd:complexType name="OtherCertStatusRefsType">
  <xsd:sequence>
    <xsd:element name="OtherRef" type="AnyType"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

The `CompleteRevocationRefs` element can contain:

- sequences of references to CRLs (`CRLRefs` element);
- sequences of references to `OCSPResponse` data as defined in RFC 2560 [9] (`OCSPRefs` element);
- other references to alternative forms of revocation data (`OtherRefs` element).

Each element in a `CRLRefs` sequence (`CrlRef` element) references one CRL. Each reference contains:

- the digest of the entire DER encoded CRL (`DigestAlgAndValue` element);

- a set of data (`CRLIdentifier` element) including the issuer (`Issuer` element), the time when the CRL was issued (`IssueTime` element) and optionally the number of the CRL (`Number` element). `CRLIdentifier` element contents MUST follow the rules established by XMLDSIG [3] in its clause 4.4.4 for strings representing Distinguished Names. In addition, this element can be dropped if the CRL could be inferred from other information. Its `URI` attribute could serve to indicate where the identified CRL is archived.

Each element in an `OCSPRefs` sequence (`OcspRef` element) references one OCSP response. Each reference contains:

- a set of data (`OCSPIdentifier` element) that includes an identifier of the responder and an indication of the time when the response was generated. The responder may be identified by its name, using the `Byname` element within `ResponderID`. It may also be identified by the digest of the server"s public key computed as mandated in RFC 2560 [9] , using the `ByKey` element. In this case the content of the `ByKey` element will be the DER value of the `byKey` field defined in RFC 2560, base64-encoded. The contents of `ByName` element MUST follow the rules established by XMLDSIG [3] in its clause 4.4.4 for strings representing Distinguished Names. The generation time indication appears in the `ProducedAt` element and corresponds to the "ProducedAt" field of the referenced response. The optional `URI` attribute could serve to indicate where the OCSP response identified is archived;

- the digest computed on the DER encoded `OCSPResponse` defined in RFC 2560 [9], appearing within `DigestAlgAndValue` element, since it MAY be needed to differentiate between two OCSP responses by the same server with their "ProducedAt" fields within the same second.

Alternative forms of validation data can be included in this property making use of the `OtherRefs` element, a sequence whose items (`OtherRef` elements) can contain any kind of information.

Should XML time-stamp tokens based in XMLDSIG be standardized and spread, this type could also serve to contain references to the full set of CRL or OCSP responses that have been used to verify the certification chain for any TSUs providing such time-stamp tokens. In this case, an element of this type could be added as an unsigned property to the XML time-stamp token using the incorporation mechanisms defined in the present specification.

### 7.4.3 The `AttributeCertificateRefs` element

This clause defines the `AttributeCertificateRefs` element that will carry the references to the full set of Attribute Authorities certificates that have been used to validate the attribute certificate.

This property MAY be used only when a user attribute certificate is present in the signature within the signature. It is an unsigned property that qualifies the signature.

An XML electronic signature aligned with the present document MAY contain at most one `AttributeCertificateRefs` element.

Below follows the schema definition for this element.

```
<xsd:element name="AttributeCertificateRefs" type="CompleteCertificateRefsType"/>
```

NOTE: Copies of the certificates referenced in this property may be held using the `AttrAuthoritiesCertValues` property.

### 7.4.4 The `AttributeRevocationRefs` element

This clause defines the `AttributeRevocationRefs` element able to carry the references to the full set of revocation data that have been used in the validation of the attribute certificate(s) present in the signature. This is an unsigned property that qualifies the signature.

This property MAY be used only when a user attribute certificate is present in the signature within the signature.

An XML electronic signature aligned with the present document MAY contain at most one `AttributeRevocationRefs` element.

Below follows the schema definition for this element.

```
<xsd:element name="AttributeRevocationRefs" type="CompleteRevocationRefsType"/>
```

NOTE: Copies of the revocation values referenced in this property may be held using the `AttributeRevocationValues` property.

## 7.5 Time-stamps on references to validation data

This clause describes the incorporation into XAdES signature of time-stamp tokens on the references to validation data defined in clause 7.4.

Electronic signatures incorporating time-stamps on validation data references are needed when there is a requirement to safeguard against the possibility of a CA key in the certificate chain ever being compromised. A verifier MAY be required to provide, on request, proof that the certification path and the revocation information used at the time of the signature were valid, even in the case where one of the issuing keys or OCSP responder keys is later compromised.

The current document defines two ways of using time-stamps to protect against this compromise:

- Time-stamp the sequence formed by the digital signature (`ds:SignatureValue` element), the `SignatureTimeStamp` element when present in the XAdES-T form, the certification path references, the Attribute Authorities certificate references and the revocation data references (for both the certificates in the certification path and in the list of Attribute Authorities certificate), when an OCSP response is used to get the status of the certificate from the signer.

- Time-stamp only the references when CRLs are used to get the status of the certificate from the signer and the status of the attribute certificates present in the signature.

The signer, verifier or both MAY obtain the time-stamp.

### 7.5.1 The `SigAndRefsTimeStamp` element

When an OCSP response is used, it is necessary to time-stamp in particular that response in the case the key from the responder would be compromised. Since the information contained in the OCSP response is user specific and time specific, an individual time-stamp is needed for every signature received. Instead of placing the time-stamp only over the certification path references and the revocation information references, which include the OCSP response, the time-stamp is placed on the digital signature (`ds:SignatureValue` element), the signature time-stamp(s) present in the XAdES-T form, the certification path references and the revocation status references. For the same cryptographic price, this will provide an integrity mechanism over the electronic signature. Any modification can be immediately detected. It should be noticed that other means of protecting/detecting the integrity of the electronic signature exist and could be used.

The `SigAndRefsTimeStamp` element is an unsigned property qualifying the signature. Clause B.1 proposes a XAdES form that can incorporate one or more `SigAndRefsTimeStamp` elements.

Below follows the schema definition for this element.

```
<xsd:element name="SigAndRefsTimeStamp" type="XAdESTimeStampType"/>
```

This property contains a time-stamp token that covers the following data objects: `ds:SignatureValue` element, all present `SignatureTimeStamp` elements, `CompleteCertificateRefs`, `CompleteRevocationRefs`, and when present, `AttributeCertificateRefs` and `AttributeRevocationRefs`.

Depending whether all the aforementioned time-stamped unsigned properties and the `SigAndRefsTimeStamp` property itself have the same parent or not, its contents may be different. Details are given in clauses below.

#### 7.5.1.1 Not distributed case

When `SigAndRefsTimeStamp` and all the unsigned properties covered by its time-stamp token have the same parent, this property uses the Implicit mechanism. The input to the computation of the digest value MUST be the result of taking in order each of the data objects listed below, canonicalize each one and concatenate the resulting octet streams:

1) The `ds:SignatureValue` element.

2) Those among the following unsigned properties that appear before `SigAndRefsTimeStamp`, in their order of appearance within the `UnsignedSignatureProperties` element:

- `SignatureTimeStamp` elements.
- The `CompleteCertificateRefs` element.
- The `CompleteRevocationRefs` element.
- The AttributeCertificateRefs element if this property is present.
- The AttributeRevocationRefs element if this property is present.

Below follows the list -in order- of data objects that contribute to the digest computation. Elements within [] contribute in their order of appearance within the `UnsignedSignatureProperties` element, not in the order they are enumerated below:

```
(ds:SignatureValue, [SignatureTimeStamp+, CompleteCertificateRefs, CompleteRevocationRefs,
AttributeCertificateRefs?, AttributeRevocationRefs?]).
```

#### 7.5.1.2 Distributed case

When `SigAndRefsTimeStamp` and some of the unsigned properties covered by its time-stamp token DO NOT have the same parent, applications MUST build this property as indicated below:

1) No `Include` element will be added for `ds:SignatureValue`. All applications MUST implicitly assume its contribution to the digest input (see below in this clause).
2) Generate one `Include` element per each unsigned property that MUST be covered by the time-stamp token in the order they appear listed below:
   - The `SignatureTimeStamp` elements.
   - The `CompleteCertificateRefs` element.
   - The `CompleteRevocationRefs` element.
   - The AttributeCertificateRefs element if this property is present.
   - The AttributeRevocationRefs element if this property is present.

Applications MUST build URI attributes following the rules stated in clause 7.1.4.3.1.

Generating applications MUST build digest computation input as indicated below:

1) Initialize the final octet stream as an empty octet stream.
2) Take the `ds:SignatureValue` element and its content. Canonicalize it and put the result in the final octet stream.
3) Take each unsigned property listed above in the order they have been listed above(this order MUST be the same as the order the `Include` elements appear in the property). For each one extract comment nodes, canonicalize and concatenate the resulting octet string to the final octet stream.

Validating applications MUST build digest computation input as indicated below:

1) Initialize the final octet stream to empty.
2) Take the ds:SignatureValue. Canonicalize it and put the result in the final octet stream.
3) Process in order each Include element present as indicated in clause 7.1.4.3.1. Concatenate the resulting octet strings to the final octet stream.

Below follows the list of the data objects that contribute to the digest computation. Super index $^e$ means that this property is referenced using explicit mechanism, i.e. that the property contains an `Include` element that references it:

(ds:SignatureValue, SignatureTimeStamp$^e$+, CompleteCertificateRefs$^e$, CompleteRevocationRefs$^e$, AttributeCertificateRefs$^e$?, AttributeRevocationRefs$^e$?).

## 7.5.2 The `RefsOnlyTimeStamp` element

Time-Stamping each ES with Complete Validation Data as defined above may not be efficient, particularly when the same set of CA certificates and CRL information is used to validate many signatures.

Time-Stamping CA certificates will stop any attacker from issuing bogus CA certificates that could be claimed to exist before the CA key was compromised. Any bogus time-stamped CA certificates will show that the certificate was created after the legitimate CA key was compromised. In the same way, time-stamping CA CRLs, will stop any attacker from issuing bogus CA CRLs which could be claimed to exist before the CA key was compromised.

Time-Stamping of commonly used certificates and CRLs can be done centrally, e.g. inside a company or by a service provider. This method reduces the amount of data the verifier has to time-stamp, for example it could reduce to just one time-stamp per day (i.e. in the case were all the signers use the same CA and the CRL applies for the whole day). The information that needs to be time-stamped is not the actual certificates and CRLs but the unambiguous references to those certificates and CRLs.

The hash sent to the TSA will be computed then over the concatenation of `CompleteCertificateRefs` and `CompleteRevocationRefs` elements.

The `RefsOnlyTimeStamp` element is an unsigned property qualifying the signature.

Clause B.3.1 proposes a XAdES form that can incorporate one or more `RefsOnlyTimeStamp` elements.

Below follows the schema definition for this element.

```
<xsd:element name="RefsOnlyTimeStamp" type="XAdESTimeStampType"/>
```

This property contains a time-stamp token that covers the following data objects: `CompleteCertificateRefs`, `CompleteRevocationRefs`, and when present, `AttributeCertificateRefs` and `AttributeRevocationRefs`.

Depending whether all the aforementioned time-stamped unsigned properties and the `SigAndRefsTimeStamp` property itself have the same parent or not, its contents may be different. Details are given in clauses below.

### 7.5.2.1 Not distributed case

When `RefsOnlyTimeStamp` and all the unsigned properties covered by its time-stamp token have the same parent, this property uses the Implicit mechanism. The input to the computation of the digest value MUST be the result of taking those of the unsigned properties listed below that appear before the `RefsOnlyTimeStamp` in their order of appearance within the `UnsignedSignatureProperties` element, canonicalize each one and concatenate the resulting octet streams:

- The `CompleteCertificateRefs` element.
- The `CompleteRevocationRefs` element.
- The AttributeCertificateRefs element if this property is present.
- The AttributeRevocationRefs element if this property is present.

Below follows the list of data objects that contribute to the digest computation:

```
([CompleteCertificateRefs, CompleteRevocationRefs, AttributeCertificateRefs?,
AttributeRevocationRefs?]).
```

### 7.5.2.2 Distributed case

When `RefsOnlyTimeStamp` and some of the unsigned properties covered by its time-stamp token DO NOT have the same parent, applications MUST build this property generating one `Include` element per each unsigned property that must be covered by the time-stamp token in the order they appear listed below:

- The `CompleteCertificateRefs` element.
- The `CompleteRevocationRefs` element.
- The AttributeCertificateRefs element if this property is present.
- The AttributeRevocationRefs element if this property is present.

Applications MUST build URI attributes following the rules stated in clause 7.1.4.3.1.

Generating applications MUST build digest computation input as indicated below:

1) Initialize the final octet stream as an empty octet stream.
2) Take each unsigned property listed above in the order they have been listed above(this order MUST be the same as the order the `Include` elements appear in the property). For each one extract comment nodes, canonicalize and concatenate the resulting octet stream to the final octet stream.

Validating applications MUST build digest computation input as indicated below:

1) Initialize the final octet stream as an empty octet stream.
2) Process in order each Include element present as indicated in clause 7.1.4.3.1. Concatenate the resulting octet stream to the final octet stream.

Below follows the list -in order- of the data objects that contribute to the digest computation. Superindex $^e$ means that this property is referenced using explicit mechanism, i.e. that the property contains a Include element that references it:

(CompleteCertificateRefs$^e$, CompleteRevocationRefs$^e$, AttributeCertificateRefs$^e$?, AttributeRevocationRefs$^e$?).

## 7.6 Properties for validation data values

This clause describes in detail those properties that allow the incorporation of validation data values to the electronic signature. Clause B. 2 proposes a XAdES form for adding these values to the electronic signature.

### 7.6.1 The `CertificateValues` Property element

A verifier will have to prove that the certification path was valid, at the time of the validation of the signature, up to a trust point according to the naming constraints and the certificate policy constraints from an optionally specified signature validation policy. It will be necessary to capture all the certificates from the certification path, starting with those from the signer and ending up with those of the certificate from one trusted root.

When dealing with long term electronic signatures, all the data used in the verification (including the certificate path) MUST be conveniently archived. In principle, the `CertificateValues` element contains the full set of certificates that have been used to validate the electronic signature, including the signer's certificate. However, it is not necessary to include one of those certificates into this property, if the certificate is already present in the `ds:KeyInfo` element of the signature.

If `CompleteCertificateRefs` and `CertificateValues` are present, all the certificates referenced in `CompleteCertificateRefs` MUST be present either in the `ds:KeyInfo` element of the signature or in the `CertificateValues` property element.

The `CertificateValues` is an unsigned property and qualifies the XML signature.

An XML electronic signature aligned with the present document MAY contain at most one `CertificateValues` element.

```
<xsd:element name="CertificateValues" type="CertificateValuesType"/>

<xsd:complexType name="CertificateValuesType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="EncapsulatedX509Certificate"
          type="EncapsulatedPKIDataType"/>
        <xsd:element name="OtherCertificate" type="AnyType"/>
    </xsd:choice>
    <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The `EncapsulatedX509Certificate` element is able to contain the base64 encoding of a DER-encoded X.509 certificate. The `OtherCertificate` element is a placeholder for potential future new formats of certificates.

Should XML time-stamp tokens based in XMLDSIG be standardized and spread, this type could also serve to contain the certification chain for any TSUs providing such time-stamp tokens, if these certificates are not already present in the time-stamp tokens themselves as part of the TSUs' signatures. In this case, an element of this type could be added as an unsigned property to the XML time-stamp token using the incorporation mechanisms defined in the present specification.

### 7.6.2 The `RevocationValues` property element

One way of dealing with long term electronic signatures (for instance for arbitration purposes), is to store and conveniently time-stamp all the revocation data used in the verification of such signatures.

Currently two major types of revocation data are managed in most of the systems, namely CRLs and responses of on-line certificate status servers, obtained through protocols designed for these purposes, like OCSP protocol.

When using CRLs to get revocation information, a verifier will have to make sure that he or she gets at the time of the first verification the appropriate certificate revocation information from the signer's CA. This should be done as soon as possible, after the grace period (clause 4.4.3.2 - note 5), to minimize the time delay between the generation and verification of the signature. This involves checking that the signer certificate serial number is not included in the CRL. The signer, the verifier or any other third party may obtain either this CRL. If obtained by the signer, then it shall be conveyed to the verifier. Additional CRLs for the CA certificates in the certificate path MUST also be checked by the verifier. It MAY be convenient to archive these CRLs within an archived electronic signature for ease of subsequent verification or arbitration.

When using OCSP to get revocation information, a verifier will have to make sure that she or he gets at the time of the first verification an OCSP response that contains the status "valid". This should be done as soon as possible after the generation of the signature, after the grace period (clause 4.4.3.2 - note 5). The signer, the verifier or any other third party MAY fetch this OCSP response. Since OCSP responses are transient and thus are not archived by any TSP including CA, it is the responsibility of every verifier to make sure that it is stored in a safe place.

The `RevocationValues` property element is used to hold the values of the revocation information which are to be shipped with the electronic signature. If `CompleteRevocationRefs` and `RevocationValues` are present, `RevocationValues` MUST contain the values of all the objects referenced in `CompleteRevocationRefs`.

This is an unsigned property that qualifies the signature.

An XML electronic signature aligned with the present document MAY contain at most one `RevocationValues` element.

Below follows the Schema definition for this element.

```
<xsd:element name="RevocationValues" type="RevocationValuesType"/>

<xsd:complexType name="RevocationValuesType">
  <xsd:sequence>
    <xsd:element name="CRLValues" type="CRLValuesType"
      minOccurs="0"/>
    <xsd:element name="OCSPValues" type="OCSPValuesType"
      minOccurs="0"/>
    <xsd:element name="OtherValues" type="OtherCertStatusValuesType"
      minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID"  use="optional"/>
</xsd:complexType>
```

Revocation information can include Certificate Revocation Lists (`CRLValues`) or responses from an online certificate status server (`OCSPValues`). Additionally a placeholder for other revocation information (`OtherValues`) is provided for future use.

```
<xsd:complexType name="CRLValuesType">
  <xsd:sequence>
     <xsd:element name="EncapsulatedCRLValue"
       type="EncapsulatedPKIDataType"
       maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Certificate Revocation Lists (`CRLValues`) consist of a sequence of at least one Certificate Revocation List. Each `EncapsulatedCRLValue` will contain the base64 encoding of a DER-encoded X.509 CRL.

```
<xsd:complexType name="OCSPValuesType">
  <xsd:sequence>
      <xsd:element name="EncapsulatedOCSPValue"
        type="EncapsulatedPKIDataType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

OCSP Responses (`OCSPValues`) consist of a sequence of at least one OCSP Response. The `EncapsulatedOCSPValue` element contains the base64 encoding of a DER-encoded `OCSPResponse` defined in RFC 2560 [9].

```
<xsd:complexType name="OtherCertStatusValuesType">
  <xsd:sequence>
    <xsd:element name="OtherValue" type="AnyType"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

The `OtherValues` element provides a placeholder for other revocation information that can be used in the future.

Should XML time-stamp tokens based in XMLDSIG be standardized and spread, this type could also serve to contain the values of revocation data including CRLs and OCSP responses for any TSUs providing such time-stamp tokens, if they are not already present in the time-stamp tokens themselves as part of the TSUs" signatures. In this case, an element of this type could be added as an unsigned property to the XML time-stamp token using the incorporation mechanisms defined in the present specification.

### 7.6.3 The `AttrAuthoritiesCertValues` element

This property contains the certificate values of the Attribute Authorities that have been used to validate the attribute certificate when present in the signature. Should any of the certificates present within `CertificateValues` property have been used for validate the attribute certificate, they do not need to appear within the `AttrAuthoritiesCertValues`.

If `AttributeCertificateRefs` and `AttrAuthoritiesCertValues` are present, `AttrAuthoritiesCertValues` and `CertificateValues` properties MUST contain all the certificates referenced in `AttributeCertificateRefs`.

Below follows the Schema definition for this element.

```
<xsd:element name="AttrAuthoritiesCertValues" type="CertificateValuesType"/>
```

### 7.6.4 The `AttributeRevocationValues` Property element

This property contains the set of revocation data that have been used to validate the attribute certificate when present in the signature. Should any of the revocation data present within `RevocationValues` property have been used for validate the attribute certificate, they do not need to appear within the `AttributeRevocationValues`.

If `AttributeRevocationRefs` and `AttributeRevocationValues` are present, `AttributeRevocationValues` and `RevocationValues` MUST contain the values of all the objects referenced in `AttributeRevocationRefs`.

Below follows the Schema definition for this element.

```
<xsd:element name="AttributeRevocationValues" type="RevocationValuesType"/>
```

## 7.7 The `ArchiveTimeStamp` element

Advances in computing increase the probability of being able to break algorithms and compromise keys. There is therefore a requirement to be able to protect electronic signatures against this possibility.

Over a period of time weaknesses may occur in the cryptographic algorithms used to create an electronic signature (e.g. due to the time available for cryptanalysis, or improvements in crypto analytical techniques). Before such weaknesses become likely, a verifier should take extra measures to maintain the validity of the electronic signature. Several techniques could be used to achieve this goal depending on the nature of the weakened cryptography. In order to simplify matters, a single technique, called Archive validation data, covering all the cases is being presented in the present document.

Archive validation data consists of the complete validation data and the complete certificate and revocation data, time-stamped together with the electronic signature. The Archive validation data is necessary if the hash function and the crypto algorithms that were used to create the signature are no longer secure. Also, if it cannot be assumed that the hash function used by the Time-Stamping Authority is secure, then nested time-stamps of archived electronic signature are required.

The potential for Trusted Service Provider (TSP) key compromise should be significantly lower than for user keys, because TSP(s) are expected to use stronger cryptography and better key protection. It can be expected that new algorithms (or old ones with greater key lengths) will be used. In such a case, a sequence of time-stamps will protect against forgery. Each time-stamp needs to be affixed before either the compromise of the signing key or of the cracking of the algorithms used by the TSA. TSAs (Time-Stamping Authorities) should have long keys and/or a "good" or different algorithm.

Nested time-stamps will also protect the verifier against key compromise or cracking the algorithm on the old electronic signatures.

The process will need to be performed and iterated before the cryptographic algorithms used for generating the previous time-stamp are no longer secure. Archive validation data MAY thus bear multiple embedded time-stamps.

The `ArchiveTimeStamp` element is an unsigned property qualifying the signature. Below follows the schema definition for this element.

```
<xsd:element name="ArchiveTimeStamp" type="XAdESTimeStampType"/>
```

Should a `CounterSignature` unsigned property be time-stamped by the `ArchiveTimeStamp`, any ulterior change of their contents (by addition of unsigned properties if the counter-signature is a XAdES signature, for instance) would make the validation of the `ArchiveTimeStamp`, and in consequence of the countersigned XAdES signature, fail. Implementers SHOULD, in consequence, not change the contents of the `CounterSignature` property once it has been time-stamped by the `ArchiveTimeStamp`. Implementors MAY, in these circumstances, to make use of the detached counter-signature mechanism specified in clause 7.2.4.1.

In addition it has to be noted that the present document allows to counter-sign a previously time-stamped countersignature with another `CounterSignature` property added to the embedding XAdES signature after the time-stamp container.

Depending whether all the unsigned properties covered by the time-stamp token and the `ArchiveTimeStamp` property itself have the same parent or not, its contents may be different. Details are given in clauses below.

### 7.7.1 Not distributed case

When ArchiveTimeStamp and all the unsigned properties covered by its time-stamp token have the same parent, this property uses the Implicit mechanism for all the time-stamped data objects. The input to the computation of the digest value MUST be built as follows:

1) Initialize the final octet stream as an empty octet stream.

2) Take all the ds:Reference elements in their order of appearance within ds:SignedInfo referencing whatever the signer wants to sign including the SignedProperties element. Process each one as indicated below:

   - Process the retrieved ds:Reference element according to the reference processing model of XMLDSIG.
   - If the result is a XML node set, canonicalize it. If ds:Canonicalization is present, the algorithm indicated by this element is used. If not, the standard canonicalization method specified by XMLDSIG is used.
   - Concatenate the resulting octets to the final octet stream.

3) Take the following XMLDSIG elements in the order they are listed below, canonicalize each one and concatenate each resulting octet stream to the final octet stream:

   - The ds:SignedInfo element.
   - The ds:SignatureValue element.
   - The ds:KeyInfo element, if present.

4) Take any of the following unsigned signature properties that appear before the current ArchiveTimeStamp in the order they appear within the UnsignedSignatureProperties, canonicalize each one and concatenate each resulting octet stream to the final octet stream:

   - Any present SignatureTimeStamp property.
   - Any present CounterSignature property.
   - The CompleteCertificateRefs and CompleteRevocationRefs properties if present.
   - The AttributeCertificateRefs and AttributeRevocationRefs properties if present.
   - Any present SigAndRefsTimeStamp and RefsOnlyTimeStamp property.
   - The CertificateValues property. This property MUST be added if it is not already present.
   - The RevocationValues property. This property MUST be added if it is not already present.
   - The AttrAuthoritiesCertValues property. This property MUST be added if not already present and the following conditions are true: there exist an attribute certificate in the signature AND a number of certificates that have been used in its validation do not appear in CertificateValues. Its content will satisfy with the rules specified in clause 7.6.3.
   - The AttributeRevocationValues property. This property MUST be added if not already present and there the following conditions are true: there exist an attribute certificate AND some revocation data that have been used in its validation do not appear in RevocationValues. Its content will satisfy with the rules specified in clause 7.6.4.
   - Any previous ArchiveTimestamp property.

5) Take any ds:Object element in the signature that is not referenced by any ds:Reference within ds:SignedInfo, except that one containing the QualifyingProperties element. Canonicalize each one and concatenate each resulting octet stream to the final octet stream. If ds:Canonicalization is present, the algorithm indicated by this element is used. If not, the standard canonicalization method specified by XMLDSIG is used.

### 7.7.2 Distributed case

When `ArchiveTimeStamp` and some of the unsigned properties covered by its time-stamp token DO NOT have the same parent, applications MUST use the explicit (Include) mechanism only for referencing the unsigned properties. Applications MUST build this property generating one `Include` element per each unsigned property that must be covered by the time-stamp token in the order they appear listed below:

1) Any present `SignatureTimeStamp` property.
2) Any present `CounterSignature` property.
3) The `CompleteCertificateRefs` property if present.
4) The `CompleteRevocationRefs` property if present.
5) The `AttributeCertificateRefs` property if present.
6) The `AttributeRevocationRefs` property if present.
7) Any present `SigAndRefsTimeStamp` property.
8) Any present `RefsOnlyTimeStamp` property.
9) The `CertificateValues` property. This property MUST be added if it is not already present.
10) `RevocationValues` property. This property MUST be added if it is not already present.
11) The `AttrAuthoritiesCertValues` property. This property MUST be added, if not already present, when the conditions mentioned in the non distributed case are met.
12) The `AttributeRevocationValues` property. This property MUST be added, if not already present, when the conditions mentioned in the non distributed case are met.
13) Any previously present `ArchiveTimestamp` property.

No Include elements are generated for any other XMLDSIG element present in the signature and listed in clause 7.5.2.1, although they are actually time-stamped as it is discussed in the next clause.

Generating applications MUST build digest computation input as for the Implicit case (clause 7.7.1) substituting step 4 by the one specified below:

1) Take the following unsigned signature properties in the order they are listed below, extract comment nodes, canonicalize each one and concatenate each resulting octet stream to the final octet stream. The order of appearance of their referencing `Include` elements in the `ArchiveTimeStamp` property MUST be identical to the order that unsigned properties are actually processed:
    - Any present `SignatureTimeStamp` property.
    - Any present `CounterSignature` property.
    - The `CompleteCertificateRefs` property if present.
    - The `CompleteRevocationRefs` property if present.
    - The `AttributeCertificateRefs` property if present.
    - The `AttributeRevocationRefs` property if present.
    - Any present `SigAndRefsTimeStamp` property.
    - Any present `RefsOnlyTimeStamp` property.
    - The `CertificateValues` property. This property MUST be added if it is not already present.
    - The `RevocationValues` property. This property MUST be added if it is not already present.

- The `AttrAuthoritiesCertValues` property. This property MUST be added, if not already present, when the conditions mentioned in the non distributed case are met.
- The `AttributeRevocationValues` property. This property MUST be added, if not already present, when the conditions mentioned in the non distributed case are met.
- Any previous `ArchiveTimestamp` property.

Verifying applications MUST build digest computation input as detailed below:

1) Initialize the final octet stream as an empty octet stream.

2) Take all the `ds:Reference` elements in their order of appearance within `ds:SignedInfo` referencing whatever the signer wants to sign including the `SignedProperties` element. Process each one as indicated below:

   a) Process the retrieved `ds:Reference` element according to the reference processing model of XMLDSIG.

   b) If the result is a XML node set, canonicalize it. If `ds:Canonicalization` is present, the algorithm indicated by this element is used. If not, the standard canonicalization method specified by XMLDSIG is used.

   c) Concatenate the resulting octets to the final octet stream.

3) Take the following XMLDSIG elements in the order they are listed below, canonicalize each one and concatenate each resulting octet stream to the final octet stream:

   d) The `ds:SignedInfo` element.

   e) The `ds:SignatureValue` element.

   f) The `ds:KeyInfo` element if this element is present.

4) Process in order each Include element present as indicated in clause 7.1.4.3.1. Concatenate the resulting octet streams to the final octet stream.

5) Take any `ds:Object` element in the signature that is not referenced by any `ds:Reference` within `ds:SignedInfo`, except that one containing the `QualifyingProperties` element. Canonicalize each one and concatenate each resulting octet stream to the final octet stream. If `ds:Canonicalization` is present, the algorithm indicated by this element is used. If not, the standard canonicalization method specified by XMLDSIG is used.

# 8 Conformance requirements

The present document defines conformance requirements for the generation of XAdES-BES and XAdES-EPES. At least one of these two forms must be implemented.

The present document defines conformance requirements for the verification of XAdES-BES and XAdES-EPES. At least one of the two forms must be implemented.

The present document only defines conformance requirements up to a XAdES electronic signature with complete validation data (XAdES-C). This means that none of the extended and archive forms of Electronic Signature, specified in annex B, need to be implemented to get conformance to the present document.

On verification the inclusion of optional signed and unsigned properties must be supported only to the extended that the signature is verifiable. The semantics of optional properties may be unsupported, unless specified otherwise by a signature policy.

## 8.1 Basic Electronic Signature (XAdES-BES)

A system supporting XAdES-BES signers according to the present document shall, at a minimum, support generation of a XML electronic signature consisting of the following components:

- The `ds:Signature` element as specified in [3].
- At least one of the following:
  - the `SigningCertificate` signed property (as defined in clause 7.2.2) incorporated to the signature as defined in clause 6.3;
  - the `ds:KeyInfo` element whose contents satisfy the restrictions specified in clause 4.4.1.

## 8.2 Explicit policy based Electronic Signature (XAdES-EPES)

A system supporting policy based signers according to the present document shall, at a minimum, support generation of XAdES-BES, plus:

- The `SignaturePolicyIdentifier` signed property (as defined in clause 7.2.3).

## 8.3 Verification using time-stamping

A system supporting verifiers according to the present document with time-stamping facilities shall, at a minimum, support:

- Verification of the mandated components of a XAdES-BES electronic signature, as defined in clause 8.1.
- `SignatureTimeStamp` unsigned property, as defined in clause 7.3.
- `CompleteCertificateRefs` unsigned property as defined in clause 7.4.1.
- `CompleteRevocationRefs` unsigned property, as defined in clause 7.4.2.
- Public Key Certificates, as defined in ITU-T Recommendation X.509 [6] (see clause 8.1).
- Either of:
  - Certificate Revocation Lists. as defined in ITU-T Recommendation X.509 [6] (see clause 8.2); or
  - On-line Certificate Status Protocol, as defined in RFC 2560 [9] (see clause 8.3).

## 8.4 Verification using secure records

A system supporting verifiers according to the present document shall, at a minimum, support:

- Verification of the mandated components of a XAdES-BES, as defined in clause 8.1.
- `CompleteCertificateRefs` unsigned property as defined in clause 7.4.1.
- `CompleteRevocationRefs` unsigned property, as defined in clause 7.4.2.
- A record must be maintained and cannot be undetectable modified, of the electronic signature and the time when the signature was first validated using the referenced certificates and revocation information.
- Public Key Certificates, as defined in ITU-T Recommendation X.509 [6] (see clause 8.1).
- Either of:
  - Certificate Revocation Lists. as defined in ITU-T Recommendation X.509 [6] (see clause 8.2); or
  - On-line Certificate Status Protocol, as defined in RFC 2560 [9] (see clause 8.3).

# Annex A (informative): Definitions

**Data Object (Content/Document) (source W3C/IETF Recommendation: "XML-Signature Si=yntax and Processing" [3])**

The actual binary/octet data being operated on (transformed, digested or signed) by an application, frequently an HTTP entity. Note that the proper noun Object designates a specific XML element. Occasionally we refer to a data object as a document or as a resource's content. The term element content is used to describe the data between XML start and end tags. The term XML document is used to describe data objects which conform to the XML specification.

**Signature (source W3C/IETF Recommendation February 2002: "XML-Signature Syntax and Processing" [3])**

Formally speaking, a value generated from the application of a private key to a message via a cryptographic algorithm such that it has the properties of signer authentication and message authentication (integrity). (However, we sometimes use the term signature generically such that it encompasses AuthenticationCode values as well, but we are careful to make the distinction when the property of AuthenticationSigner is relevant to the exposition.) A signature may be (non-exclusively) described as detached, enveloping or enveloped.

**Transform (source W3C/IETF Recommendation February 2002: "XML-Signature Syntax and Processing" [3])**

The processing of a data from its source to its derived form. Typical transforms include XML Canonicalization, XPath and XSLT.

**Attribute Certificate (source ITU-T Recommendation X.509: "DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS.DIRECTORY" [6])**

A set of attributes of a user together with some other information, rendered unforgeable by the digital signature created using the private key of the certification authority which issued it.

# Annex B (informative): Extended electronic signature forms

The XAdES forms specified in clause 4.4 can be extended by addition of certain unsigned properties that are defined in the present document. These properties are applicable for very long term verification, and for preventing some disaster situations which have been identified in the normative part of the present document. The clauses below give an overview of the various forms of extended signature formats in the present document.

## B.1 Extended signatures with time forms (XAdES-X)

Extended signatures with time indication forms (XAdES-X) in accordance with the present document build on signatures containing `CompleteCertificateRefs` and `CompleteRevocationRefs` properties, by adding one or more time-stamps unsigned properties.

Depending of what is time-stamped, there are two different types of XAdES-X signatures, namely, XAdES-X type 1 and XAdES-X type 2. Time-stamps in both types cover, among other elements, `CompleteCertificateRefs` and `CompleteRevocationRefs` properties. Time-stamps provide an integrity and trusted time protection over everything that is time-stamped. They protect the referenced certificates, CRLs and OCSP responses in case of a later compromise of a CA key, CRL key or OCSP issuer key.

XAdES-X type 1 is built by adding one or more `SigAndRefsTimeStamp` properties each containing one time-stamp obtained from different TSAs. These time-stamps are computed on the `SignatureValue` element, `SignatureTimeStamp` if present, `CompleteCertificateRefs` and `CompleteRevocationRefs` properties.

XAdES-X type 2 is built by adding one or more `RefsOnlyTimeStamp` properties each containing one time-stamp obtained from different TSAs. These time-stamps are computed on the `CompleteCertificateRefs` and `CompleteRevocationRefs` properties.

Below follows the most complete XAdES-X structure, which builds on a XAdES-C signature.

```
                               XMLDISG
                                  |
<ds:Signature ID?>- - - - - - - - +- - - - - - +-+-+-+
  <ds:SignedInfo>                 |            | | | |
    <ds:CanonicalizationMethod/>  |            | | | |
    <ds:SignatureMethod/>         |            | | | |
    (<ds:Reference URI? >         |            | | | |
      (<ds:Transforms/>)?         |            | | | |
      <ds:DigestMethod/>          |            | | | |
      <ds:DigestValue/>           |            | | | |
    </ds:Reference>)+             |            | | | |
  </ds:SignedInfo/>               |            | | | |
  <ds:SignatureValue>             |            | | | |
  (<ds:KeyInfo>)? - - - - - - - - +            | | | |
                                               | | | |
  <ds:Object>                                  | | | |
                                               | | | |
    <QualifyingProperties>                     | | | |
                                               | | | |
      <SignedProperties>                       | | | |
                                               | | | |
        <SignedSignatureProperties>            | | | |
          (SigningTime)?                       | | | |
          (SigningCertificate)?                | | | |
          (SignaturePolicyIdentifier)?         | | | |
          (SignatureProductionPlace)?          | | | |
          (SignerRole)?                        | | | |
        </SignedSignatureProperties>           | | | |
                                               | | | |
        <SignedDataObjectProperties>           | | | |
          (DataObjectFormat)*                  | | | |
          (CommitmentTypeIndication)*          | | | |
          (AllDataObjectsTimeStamp)*           | | | |
          (IndividualDataObjectsTimeStamp)*    | | | |
        </SignedDataObjectPropertiesSigned>    | | | |
                                               | | | |
      </SignedProperties>                      | | | |
                                               | | | |
      <UnsignedProperties>                     | | | |
                                               | | | |
        <UnsignedSignatureProperties>          | | | |
          (CounterSignature)*- - - - - - - - - + | | |
          (SignatureTimeStamp)*- - - - - - - -   + | |
          (CompleteCertificateRefs)                | |
          (CompleteRevocationRefs)                 | |
          (AttributeCertificateRefs)?              | |
          (AttributeRevocationRefs)? - - - - - - - + |
          ((SigAndRefsTimeStamp)* |                  |
          (RefsOnlyTimeStamp)*)                      |
        </UnsignedSignatureProperties>- - - - -+-+-+ |
                                               | | | |
      </UnsignedProperties>                    | | | |
                                               | | | |
    </QualifyingProperties>                    | | | |
                                               | | | |
  </ds:Object>                                 | | | |
</ds:Signature>- - - - - - - - - - - - - - - - +-+-+-+
                                               | | | |
                                 XAdES-BES(-EPES)| | |
                                                 | | |
                                           XAdES-T | |
                                                   | |
                                             XAdES-C |
                                                     |
                                               XAdES-X
```

## B.2 Extended long electronic signatures with time (XAdES-X-L)

Extended long electronic signatures with time (XAdES-X-L) forms in accordance with the present document build up on XAdES-X types 1 or 2 by adding the `CertificateValues` and `RevocationValues` unsigned properties aforementioned.

The structure for the most complete XAdES-X-L, built on the most complete XAdES-X signature, is shown below.

```
                                XMLDISG
                                  |
<ds:Signature ID?>- - - - - - - - +- - - - - +-+-+-+-+
  <ds:SignedInfo>                 |          | | | | |
    <ds:CanonicalizationMethod/>  |          | | | | |
    <ds:SignatureMethod/>         |          | | | | |
    (<ds:Reference URI? >         |          | | | | |
      (<ds:Transforms/>)?         |          | | | | |
      <ds:DigestMethod/>          |          | | | | |
      <ds:DigestValue/>           |          | | | | |
    </ds:Reference>)+             |          | | | | |
  </ds:SignedInfo>                |          | | | | |
  <ds:SignatureValue/>            |          | | | | |
  (<ds:KeyInfo>)?  - - - - - - - -+          | | | | |
                                             | | | | |
  <ds:Object>                                | | | | |
                                             | | | | |
    <QualifyingProperties>                   | | | | |
                                             | | | | |
      <SignedProperties>                     | | | | |
                                             | | | | |
        <SignedSignatureProperties>          | | | | |
          (SigningTime)?                     | | | | |
          (SigningCertificate)?              | | | | |
          (SignaturePolicyIdentifier)?       | | | | |
          (SignatureProductionPlace)?        | | | | |
          (SignerRole)?                      | | | | |
        </SignedSignatureProperties>         | | | | |
                                             | | | | |
        <SignedDataObjectProperties>         | | | | |
          (DataObjectFormat)*                | | | | |
          (CommitmentTypeIndication)*        | | | | |
          (AllDataObjectsTimeStamp)*         | | | | |
          (IndividualDataObjectsTimeStamp)*  | | | | |
        </SignedDataObjectPropertiesSigned>  | | | | |
                                             | | | | |
      </SignedProperties>                    | | | | |
                                             | | | | |
      <UnsignedProperties>                   | | | | |
                                             | | | | |
        <UnsignedSignatureProperties>        | | | | |
          (CounterSignature)*- - - - - - - - + | | | |
          (SignatureTimeStamp)*- - - - - - - - + | | |
          (CompleteCertificateRefs)              | | |
          (CompleteRevocationRefs)               | | |
          (AttributeCertificateRefs)?            | | |
          (AttributeRevocationRefs)? - - - - - - + | |
          ((SigAndRefsTimeStamp)*  |               | |
          (RefsOnlyTimeStamp)*)- - - - - - - - - - + |
          (CertificatesValues)                       |
          (RevocationValues)                         |
          (AttrAuthoritiesCertValues)?               |
          (AttributeRevocationValues)?               |
        </UnsignedSignatureProperties>- - - -+-+-+-+ |
                                             | | | | |
      </UnsignedProperties>                  | | | | |
                                             | | | | |
    </QualifyingProperties>                  | | | | |
                                             | | | | |
  </ds:Object>                               | | | | |
</ds:Signature>- - - - - - - - - - - - - - - +-+-+-+-+
                                             | | | | |
                               XAdES-BES(-EPES)| | | |
                                               | | | |
                                         XAdES-T | | |
                                                 | | |
                                           XAdES-C | |
                                                   | |
                                             XAdES-X |
                                                     |
                                              XAdES-X-L
```

## B.3 Archival electronic signatures (XAdES-A)

Archival signatures in accordance with the present document MUST incorporate `CertificateValues`, `RevocationValues` and one or more `ArchiveTimeStamp` unsigned properties. They MAY contain other properties. Each `ArchiveTimeStamp` element contains a time-stamp token covering among other elements, those ones that contain validation data (see clause 7.7). These forms are used for archival of signatures. Successive time-stamps protect the whole material against vulnerable hashing algorithms or the breaking of the cryptographic material or algorithms.

Below follows the structure of a XAdES-A built on a XAdES-X-L, as an example of the most complete archival form.

```
                                XMLDISG
                                  |
<ds:Signature ID?>- - - - - - - - +- - - - - +-+-+-+-+-+
  <ds:SignedInfo>                 |          | | | | | |
    <ds:CanonicalizationMethod/>  |          | | | | | |
    <ds:SignatureMethod/>         |          | | | | | |
    (<ds:Reference (URI=)? >      |          | | | | | |
      (<ds:Transforms/>)?         |          | | | | | |
      <ds:DigestMethod/>          |          | | | | | |
      <ds:DigestValue/>           |          | | | | | |
    </ds:Reference>)+             |          | | | | | |
  </ds:SignedInfo>                |          | | | | | |
  <ds:SignatureValue/>            |          | | | | | |
  (<ds:KeyInfo>)? - - - - - - - - +          | | | | | |
  <ds:Object>                                | | | | | |
                                             | | | | | |
    <QualifyingProperties>                   | | | | | |
                                             | | | | | |
      <SignedProperties>                     | | | | | |
                                             | | | | | |
        <SignedSignatureProperties>          | | | | | |
          (SigningTime)?                     | | | | | |
          (SigningCertificate)?              | | | | | |
          (SignaturePolicyIdentifier)?       | | | | | |
          (SignatureProductionPlace)?        | | | | | |
          (SignerRole)?                      | | | | | |
        </SignedSignatureProperties>         | | | | | |
                                             | | | | | |
        <SignedDataObjectProperties>         | | | | | |
          (DataObjectFormat)*                | | | | | |
          (CommitmentTypeIndication)*        | | | | | |
          (AllDataObjectsTimeStamp)*         | | | | | |
          (IndividualDataObjectsTimeStamp)*  | | | | | |
        </SignedDataObjectPropertiesSigned>  | | | | | |
                                             | | | | | |
      </SignedProperties>                    | | | | | |
                                             | | | | | |
      <UnsignedProperties>                   | | | | | |
                                             | | | | | |
        <UnsignedSignatureProperties>        | | | | | |
          (CounterSignature)*- - - - - - - - + | | | | |
          (SignatureTimeStamp)*- - - - - - - - + | | | |
          (CompleteCertificateRefs)?             | | | |
          (CompleteRevocationRefs)?              | | | |
          (AttributeCertificateRefs)?            | | | |
          (AttributeRevocationRefs)? - - - - - - + | | |
          ((SigAndRefsTimeStamp)*  |               | | |
          (RefsOnlyTimeStamp)*)? - - - - - - - - - + | |
          (CertificatesValues)                       | |
          (RevocationValues                          | |
          (AttrAuthoritiesCertValues)?               | |
          (AttributeRevocationValues)?- - - - - - - -+ |
          (ArchiveTimeStamp)+                          |
        </UnsignedSignatureProperties>- - - -+-+-+-+-+ |
                                             | | | | | |
      </UnsignedProperties>                  | | | | | |
                                             | | | | | |
    </QualifyingProperties>                  | | | | | |
                                             | | | | | |
  </ds:Object>                               | | | | | |
                                             | | | | | |
</ds:Signature>- - - - - - - - - - - - - - - +-+-+-+-+-+
                                             | | | | | |
                             XAdES-BES(-EPES)| | | | | |
                                               | | | | |
                                         XAdES-T | | | |
                                                 | | | |
                                           XAdES-C | | |
                                                   | | |
                                             XAdES-X | |
                                                     | |
                                             XAdES-X-L |
                                                       |
                                                 XAdES-A
```

# Annex C (informative): concepts and rationales

This annex presents details on some of the concepts used in the standard. It also provides the rationale for certain normative parts of the present document.

## C.1 Multiple signatures and countersignatures

Some electronic signatures may only be valid if they bear more than one signature. This is generally the case, for example, when a contract is signed between two parties. The ordering of the signatures may or may not be important, i.e. one may or may not need to be applied before the other. This allows establishing two basic categories for multiple signatures:

- independent signatures;
- countersignatures.

Independent signatures are parallel signatures where the ordering of the signatures is not important. The computation of these signatures is performed on exactly the same input but using different private keys.

Countersignatures are signatures that are applied one after the other and are used where the order the signatures are applied is important. In these situations the first signature signs the signed data object. Each additional signature may sign in turn the latest previously generated signature, or all the previously generated signatures and the signed document.

The referencing mechanism present in XMLDSIG gives full support to countersignatures. Using them, the countersignatures may be placed and kept in different ways: they may be embedded one within the other, or they may be detached from the rest as long as their corresponding `<ds:Reference>` elements ensure that each signature actually signs the previously generated signature (or all the previously generated signatures and the signed document if this is the requirement). While XMLDSIG supports these features, it does not propose any standard format for countersignatures as it considers this topic being out of its scope.

The present document defines a new URI value, which, when assigned as value of the `Type` attribute of a `ds:Reference` element, denotes that the enclosing XAdES signature is in fact, a countersignature of another signature.

In addition the present document defines with the `CounterSignature` property a standard way of managing countersignatures that:

- are computed on the values of the latest previously generated signatures;
- are embedded within the signatures that they countersign so that the first electronic signature (the one computed on the data objects actually signed) contains all the additional countersignatures that have to be verified.

This proposal does not, of course, satisfy all the potential requirements that real situations may pose in terms of relationships among electronic signatures and documents. This would require more complexity and likely the need to define new XML containers for the signatures, which is currently out of scope of XAdES.

Independent signatures will not appear as `CounterSignature` properties of another independent one.

# Annex D (normative): Schema definitions

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://uri.etsi.org/01903/v1.3.2#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://uri.etsi.org/01903/v1.3.2#"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" elementFormDefault="qualified">

  <xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd"/>

<!-- Start auxiliary types definitions: AnyType, ObjectIdentifierType,
EncapsulatedPKIDataType and containers for time-stamp tokens -->

<!-- Start AnyType -->

  <xsd:element name="Any" type="AnyType"/>
  <xsd:complexType name="AnyType" mixed="true">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:any namespace="##any" processContents="lax"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##any"/>
  </xsd:complexType>

<!-- End AnyType -->

<!-- Start ObjectIdentifierType-->

  <xsd:element name="ObjectIdentifier" type="ObjectIdentifierType"/>
  <xsd:complexType name="ObjectIdentifierType">
    <xsd:sequence>
      <xsd:element name="Identifier" type="IdentifierType"/>
      <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
      <xsd:element name="DocumentationReferences"
        type="DocumentationReferencesType" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="IdentifierType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:anyURI">
        <xsd:attribute name="Qualifier" type="QualifierType" use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:simpleType name="QualifierType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="OIDAsURI"/>
      <xsd:enumeration value="OIDAsURN"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="DocumentationReferencesType">
    <xsd:sequence maxOccurs="unbounded">
      <xsd:element name="DocumentationReference" type="xsd:anyURI"/>
    </xsd:sequence>
  </xsd:complexType>

<!-- End ObjectIdentifierType-->

<!-- Start EncapsulatedPKIDataType-->

  <xsd:element name="EncapsulatedPKIData" type="EncapsulatedPKIDataType"/>
  <xsd:complexType name="EncapsulatedPKIDataType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:base64Binary">
        <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
        <xsd:attribute name="Encoding" type="xsd:anyURI" use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

<!-- End EncapsulatedPKIDataType -->

  <!-- Start time-stamp containers types -->

  <!-- Start GenericTimeStampType -->
```

```
<xsd:element name="Include" type="IncludeType"/>
<xsd:complexType name="IncludeType">
  <xsd:attribute name="URI" type="xsd:anyURI" use="required"/>
  <xsd:attribute name="referencedData" type="xsd:boolean" use="optional"/>
</xsd:complexType>
<xsd:element name="ReferenceInfo" type="ReferenceInfoType"/>
<xsd:complexType name="ReferenceInfoType">
  <xsd:sequence>
    <xsd:element ref="ds:DigestMethod"/>
    <xsd:element ref="ds:DigestValue"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
  <xsd:attribute name="URI" type="xsd:anyURI" use="optional"/>
</xsd:complexType>

<xsd:complexType name="GenericTimeStampType" abstract="true">
  <xsd:sequence>
    <xsd:choice minOccurs="0">
      <xsd:element ref="Include" maxOccurs="unbounded"/>
      <xsd:element ref="ReferenceInfo" maxOccurs="unbounded"/>
    </xsd:choice>
    <xsd:element ref="ds:CanonicalizationMethod" minOccurs="0"/>
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="EncapsulatedTimeStamp"
         type="EncapsulatedPKIDataType"/>
      <xsd:element name="XMLTimeStamp" type="AnyType"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>

<!-- End GenericTimeStampType -->

<!-- Start XAdESTimeStampType"/>

<xsd:element name="XAdESTimeStamp" type="XAdESTimeStampType"/>
<xsd:complexType name="XAdESTimeStampType">
  <xsd:complexContent>
    <xsd:restriction base="GenericTimeStampType">
      <xsd:sequence>
        <xsd:element ref="Include" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="ds:CanonicalizationMethod" minOccurs="0"/>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="EncapsulatedTimeStamp"
              type="EncapsulatedPKIDataType"/>
          <xsd:element name="XMLTimeStamp" type="AnyType"/>
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

<!-- End XAdESTimeStampType -->

<!-- Start OtherTimeStampType"/>

<xsd:element name="OtherTimeStamp" type="OtherTimeStampType"/>
<xsd:complexType name="OtherTimeStampType">
  <xsd:complexContent>
    <xsd:restriction base="GenericTimeStampType">
      <xsd:sequence>
        <xsd:element ref="ReferenceInfo"  maxOccurs="unbounded"/>
        <xsd:element ref="ds:CanonicalizationMethod" minOccurs="0"/>
        <xsd:choice>
            <xsd:element name="EncapsulatedTimeStamp"
              type="EncapsulatedPKIDataType"/>
            <xsd:element name="XMLTimeStamp" type="AnyType"/>
          </xsd:choice>
        </xsd:sequence>
      <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

<!-- End OtherTimeStampType -->

<!-- End time-stamp containers types -->
```

```
<!-- End auxiliary types definitions-->

<!-- Start container types -->

<!-- Start QualifyingProperties -->

  <xsd:element name="QualifyingProperties" type="QualifyingPropertiesType"/>

  <xsd:complexType name="QualifyingPropertiesType">
    <xsd:sequence>
      <xsd:element name="SignedProperties" type="SignedPropertiesType"
        minOccurs="0"/>
      <xsd:element name="UnsignedProperties" type="UnsignedPropertiesType"
        minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="Target" type="xsd:anyURI" use="required"/>
    <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
  </xsd:complexType>

<!-- End QualifyingProperties -->

<!-- Start SignedProperties-->

  <xsd:element name="SignedProperties" type="SignedPropertiesType"/>

  <xsd:complexType name="SignedPropertiesType">
    <xsd:sequence>
      <xsd:element name="SignedSignatureProperties"
        type="SignedSignaturePropertiesType"/>
      <xsd:element name="SignedDataObjectProperties"
        type="SignedDataObjectPropertiesType" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
  </xsd:complexType>

<!-- End SignedProperties-->

<!-- Start UnsignedProperties-->

<xsd:element name="UnsignedProperties" type="UnsignedPropertiesType" />

  <xsd:complexType name="UnsignedPropertiesType">
    <xsd:sequence>
      <xsd:element name="UnsignedSignatureProperties"
        type="UnsignedSignaturePropertiesType" minOccurs="0"/>
        <xsd:element name="UnsignedDataObjectProperties"
          type="UnsignedDataObjectPropertiesType" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
  </xsd:complexType>

<!-- End UnsignedProperties-->

<!-- Start SignedSignatureProperties-->

<xsd:element name="SignedSignatureProperties"
  type="SignedSignaturePropertiesType" />

<xsd:complexType name="SignedSignaturePropertiesType">
  <xsd:sequence>
    <xsd:element name="SigningTime" type="xsd:dateTime" minOccurs="0"/>
    <xsd:element name="SigningCertificate" type="CertIDListType"
      minOccurs="0"/>
    <xsd:element name="SignaturePolicyIdentifier"
     type="SignaturePolicyIdentifierType" minOccurs="0"/>
    <xsd:element name="SignatureProductionPlace"
      type="SignatureProductionPlaceType"
     minOccurs="0"/>
    <xsd:element name="SignerRole" type="SignerRoleType" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>

<!-- End SignedSignatureProperties-->


<!-- Start SignedDataObjectProperties-->
```

```
<xsd:element name="SignedDataObjectProperties"
  type="SignedDataObjectPropertiesType"/>

<xsd:complexType name="SignedDataObjectPropertiesType">
  <xsd:sequence>
    <xsd:element name="DataObjectFormat" type="DataObjectFormatType"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="CommitmentTypeIndication"
      type="CommitmentTypeIndicationType" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="AllDataObjectsTimeStamp" type="XAdESTimeStampType"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="IndividualDataObjectsTimeStamp"
      type="XAdESTimeStampType"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>

<!-- End SignedDataObjectProperties-->


<!-- Start UnsignedSignatureProperties-->

<xsd:element name="UnsignedSignatureProperties"
  type="UnsignedSignaturePropertiesType"/>

<xsd:complexType name="UnsignedSignaturePropertiesType">
  <xsd:choice maxOccurs="unbounded">
    <xsd:element name="CounterSignature" type="CounterSignatureType"/>
    <xsd:element name="SignatureTimeStamp" type="XAdESTimeStampType"/>
    <xsd:element name="CompleteCertificateRefs"
      type="CompleteCertificateRefsType"/>
    <xsd:element name="CompleteRevocationRefs"
      type="CompleteRevocationRefsType"/>
    <xsd:element name="AttributeCertificateRefs"
      type="CompleteCertificateRefsType"/>
    <xsd:element name="AttributeRevocationRefs"
      type="CompleteRevocationRefsType"/>
    <xsd:element name="SigAndRefsTimeStamp" type="XAdESTimeStampType"/>
    <xsd:element name="RefsOnlyTimeStamp" type="XAdESTimeStampType"/>
    <xsd:element name="CertificateValues"
      type="CertificateValuesType"/>
    <xsd:element name="RevocationValues" type="RevocationValuesType"/>
    <xsd:element name="AttrAuthoritiesCertValues"
      type="CertificateValuesType"/>
    <xsd:element name="AttributeRevocationValues"
      type="RevocationValuesType"/>
    <xsd:element name="ArchiveTimeStamp" type="XAdESTimeStampType"/>
    <xsd:any namespace="##other" />
  </xsd:choice>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>


<!-- End UnsignedSignatureProperties-->


<!-- Start UnsignedDataObjectProperties-->

<xsd:element name="UnsignedDataObjectProperties"
  type="UnsignedDataObjectPropertiesType" />

<xsd:complexType name="UnsignedDataObjectPropertiesType">
  <xsd:sequence>
    <xsd:element name="UnsignedDataObjectProperty" type="AnyType"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>

<!-- End UnsignedDataObjectProperties-->

<!-- Start QualifyingPropertiesReference-->

<xsd:element name="QualifyingPropertiesReference"
  type="QualifyingPropertiesReferenceType"/>

<xsd:complexType name="QualifyingPropertiesReferenceType">
```

```
  <xsd:attribute name="URI" type="xsd:anyURI" use="required"/>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>

<!-- End QualifyingPropertiesReference-->

<!-- End container types -->


<!-- Start SigningTime element -->
  <xsd:element name="SigningTime" type="xsd:dateTime"/>

<!-- End SigningTime element -->

<!-- Start SigningCertificate -->
  <xsd:element name="SigningCertificate" type="CertIDListType"/>
  <xsd:complexType name="CertIDListType">
    <xsd:sequence>
      <xsd:element name="Cert" type="CertIDType" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="CertIDType">
    <xsd:sequence>
      <xsd:element name="CertDigest" type="DigestAlgAndValueType"/>
      <xsd:element name="IssuerSerial" type="ds:X509IssuerSerialType"/>
    </xsd:sequence>
    <xsd:attribute name="URI" type="xsd:anyURI" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="DigestAlgAndValueType">
    <xsd:sequence>
      <xsd:element ref="ds:DigestMethod"/>
      <xsd:element ref="ds:DigestValue"/>
    </xsd:sequence>
  </xsd:complexType>

<!-- End SigningCertificate -->

<!-- Start SignaturePolicyIdentifier -->

  <xsd:element name="SignaturePolicyIdentifier"
    type="SignaturePolicyIdentifierType"/>
  <xsd:complexType name="SignaturePolicyIdentifierType">
    <xsd:choice>
      <xsd:element name="SignaturePolicyId" type="SignaturePolicyIdType"/>
      <xsd:element name="SignaturePolicyImplied"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="SignaturePolicyIdType">
    <xsd:sequence>
      <xsd:element name="SigPolicyId" type="ObjectIdentifierType"/>
      <xsd:element ref="ds:Transforms" minOccurs="0"/>
      <xsd:element name="SigPolicyHash" type="DigestAlgAndValueType"/>
      <xsd:element name="SigPolicyQualifiers"
        type="SigPolicyQualifiersListType" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SigPolicyQualifiersListType">
    <xsd:sequence>
      <xsd:element name="SigPolicyQualifier" type="AnyType"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="SPURI" type="xsd:anyURI"/>
  <xsd:element name="SPUserNotice" type="SPUserNoticeType"/>
  <xsd:complexType name="SPUserNoticeType">
    <xsd:sequence>
      <xsd:element name="NoticeRef" type="NoticeReferenceType"
        minOccurs="0"/>
      <xsd:element name="ExplicitText" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="NoticeReferenceType">
    <xsd:sequence>
      <xsd:element name="Organization" type="xsd:string"/>
      <xsd:element name="NoticeNumbers" type="IntegerListType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="IntegerListType">
    <xsd:sequence>
```

```
      <xsd:element name="int" type="xsd:integer" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

<!-- End SignaturePolicyIdentifier -->


<!-- Start CounterSignature -->
  <xsd:element name="CounterSignature" type="CounterSignatureType"/>
  <xsd:complexType name="CounterSignatureType">
    <xsd:sequence>
      <xsd:element ref="ds:Signature"/>
    </xsd:sequence>
  </xsd:complexType>

<!-- End CounterSignature -->

<!-- Start DataObjectFormat -->

  <xsd:element name="DataObjectFormat" type="DataObjectFormatType"/>
  <xsd:complexType name="DataObjectFormatType">
    <xsd:sequence>
      <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
      <xsd:element name="ObjectIdentifier" type="ObjectIdentifierType"
        minOccurs="0"/>
      <xsd:element name="MimeType" type="xsd:string" minOccurs="0"/>
      <xsd:element name="Encoding" type="xsd:anyURI" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="ObjectReference" type="xsd:anyURI" use="required"/>
  </xsd:complexType>

<!-- End DataObjectFormat -->

<!-- Start CommitmentTypeIndication -->

  <xsd:element name="CommitmentTypeIndication" type="CommitmentTypeIndicationType"/>
  <xsd:complexType name="CommitmentTypeIndicationType">
    <xsd:sequence>
      <xsd:element name="CommitmentTypeId" type="ObjectIdentifierType"/>
      <xsd:choice>
        <xsd:element name="ObjectReference" type="xsd:anyURI"
          maxOccurs="unbounded"/>
        <xsd:element name="AllSignedDataObjects"/>
      </xsd:choice>
      <xsd:element name="CommitmentTypeQualifiers"
        type="CommitmentTypeQualifiersListType" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="CommitmentTypeQualifiersListType">
    <xsd:sequence>
      <xsd:element name="CommitmentTypeQualifier" type="AnyType"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

<!-- End CommitmentTypeIndication -->

<!-- Start SignatureProductionPlace -->

  <xsd:element name="SignatureProductionPlace"
    type="SignatureProductionPlaceType"/>
  <xsd:complexType name="SignatureProductionPlaceType">
    <xsd:sequence>
      <xsd:element name="City" type="xsd:string" minOccurs="0"/>
      <xsd:element name="StateOrProvince" type="xsd:string" minOccurs="0"/>
      <xsd:element name="PostalCode" type="xsd:string" minOccurs="0"/>
      <xsd:element name="CountryName" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>

<!-- End SignatureProductionPlace -->

<!-- Start SignerRole -->

<xsd:element name="SignerRole" type="SignerRoleType"/>
<xsd:complexType name="SignerRoleType">
  <xsd:sequence>
    <xsd:element name="ClaimedRoles" type="ClaimedRolesListType"
```

```
      minOccurs="0"/>
    <xsd:element name="CertifiedRoles" type="CertifiedRolesListType"
      minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ClaimedRolesListType">
  <xsd:sequence>
    <xsd:element name="ClaimedRole" type="AnyType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CertifiedRolesListType">
  <xsd:sequence>
    <xsd:element name="CertifiedRole" type="EncapsulatedPKIDataType"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<!-- End SignerRole -->


  <xsd:element name="AllDataObjectsTimeStamp" type="XAdESTimeStampType"/>

  <xsd:element name="IndividualDataObjectsTimeStamp"
    type="XAdESTimeStampType"/>

  <xsd:element name="SignatureTimeStamp" type="XAdESTimeStampType"/>

<!-- Start CompleteCertificateRefs -->

<xsd:element name="CompleteCertificateRefs"
type="CompleteCertificateRefsType"/>

<xsd:complexType name="CompleteCertificateRefsType">
  <xsd:sequence>
    <xsd:element name="CertRefs" type="CertIDListType" />
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>

<!-- End CompleteCertificateRefs -->


<!-- Start CompleteRevocationRefs-->

<xsd:element name="CompleteRevocationRefs"
type="CompleteRevocationRefsType"/>

<xsd:complexType name="CompleteRevocationRefsType">
  <xsd:sequence>
    <xsd:element name="CRLRefs" type="CRLRefsType" minOccurs="0"/>
    <xsd:element name="OCSPRefs" type="OCSPRefsType" minOccurs="0"/>
    <xsd:element name="OtherRefs" type="OtherCertStatusRefsType"
      minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>

<xsd:complexType name="CRLRefsType">
  <xsd:sequence>
    <xsd:element name="CRLRef" type="CRLRefType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CRLRefType">
  <xsd:sequence>
    <xsd:element name="DigestAlgAndValue" type="DigestAlgAndValueType"/>
    <xsd:element name="CRLIdentifier" type="CRLIdentifierType"
      minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CRLIdentifierType">
  <xsd:sequence>
    <xsd:element name="Issuer" type="xsd:string"/>
    <xsd:element name="IssueTime" type="xsd:dateTime" />
    <xsd:element name="Number" type="xsd:integer"  minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="URI" type="xsd:anyURI" use="optional"/>
</xsd:complexType>

<xsd:complexType name="OCSPRefsType">
  <xsd:sequence>
    <xsd:element name="OCSPRef" type="OCSPRefType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="OCSPRefType">
  <xsd:sequence>
    <xsd:element name="OCSPIdentifier" type="OCSPIdentifierType"/>
    <xsd:element name="DigestAlgAndValue" type="DigestAlgAndValueType"
      minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

  <xsd:complexType name="ResponderIDType">
    <xsd:choice>
      <xsd:element name="ByName" type="xsd:string"/>
      <xsd:element name="ByKey" type="xsd:base64Binary"/>
    </xsd:choice>
  </xsd:complexType>

<xsd:complexType name="OCSPIdentifierType">
  <xsd:sequence>
    <xsd:element name="ResponderID" type="ResponderIDType"/>
    <xsd:element name="ProducedAt" type="xsd:dateTime"/>
  </xsd:sequence>
  <xsd:attribute name="URI" type="xsd:anyURI" use="optional"/>
</xsd:complexType>

<xsd:complexType name="OtherCertStatusRefsType">
  <xsd:sequence>
    <xsd:element name="OtherRef" type="AnyType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<!-- End CompleteRevocationRefs-->

<xsd:element name="AttributeCertificateRefs"
  type="CompleteCertificateRefsType"/>

<xsd:element name="AttributeRevocationRefs"
  type="CompleteRevocationRefsType"/>
<xsd:element name="SigAndRefsTimeStamp" type="XAdESTimeStampType"/>

<xsd:element name="RefsOnlyTimeStamp" type="XAdESTimeStampType"/>

<!-- Start CertificateValues -->

<xsd:element name="CertificateValues" type="CertificateValuesType"/>

<xsd:complexType name="CertificateValuesType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="EncapsulatedX509Certificate"
      type="EncapsulatedPKIDataType"/>
     <xsd:element name="OtherCertificate" type="AnyType"/>
    </xsd:choice>
    <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>

<!-- End CertificateValues -->

<!-- Start RevocationValues-->

<xsd:element name="RevocationValues" type="RevocationValuesType"/>

<xsd:complexType name="RevocationValuesType">
  <xsd:sequence>
    <xsd:element name="CRLValues" type="CRLValuesType" minOccurs="0"/>
    <xsd:element name="OCSPValues" type="OCSPValuesType" minOccurs="0"/>
    <xsd:element name="OtherValues" type="OtherCertStatusValuesType" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
</xsd:complexType>

<xsd:complexType name="CRLValuesType">
  <xsd:sequence>
```

```
     <xsd:element name="EncapsulatedCRLValue" type="EncapsulatedPKIDataType"
       maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="OCSPValuesType">
  <xsd:sequence>
      <xsd:element name="EncapsulatedOCSPValue"
        type="EncapsulatedPKIDataType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="OtherCertStatusValuesType">
  <xsd:sequence>
    <xsd:element name="OtherValue" type="AnyType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<!-- End RevocationValues-->

<xsd:element name="AttrAuthoritiesCertValues" type="CertificateValuesType"/>

<xsd:element name="AttributeRevocationValues" type="RevocationValuesType"/>

<xsd:element name="ArchiveTimeStamp" type="XAdESTimeStampType"/>


</xsd:schema>
```

# Annex E (informative): DTD

```
<?xml version="1.0" encoding="UTF-8"?>

<!ENTITY % Any.ANY ' '>
<!ENTITY % XMLTimeStamp.ANY ' '>
<!ENTITY % Method.ANY ' '>

<!-- Start Any -->

<!ELEMENT Any (#PCDATA   %Any.ANY;)*>

<!-- End Any -->

<!-- Start ObjectIdentifier -->

<!ELEMENT ObjectIdentifier (Identifier, Description?,
DocumentationReferences?)>
<!ELEMENT Identifier (#PCDATA)>
<!ATTLIST Identifier
    Qualifier (OIDAsURI | OIDAsURN) #IMPLIED
>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT DocumentationReferences (DocumentationReference)+>
<!ELEMENT DocumentationReference (#PCDATA)>

<!-- End ObjectIdentifier -->

<!-- Start EncapsulatedPKIData -->

<!ELEMENT EncapsulatedPKIData (#PCDATA)>
<!ATTLIST EncapsulatedPKIData
    Id ID #IMPLIED
    Encoding CDATA #IMPLIED
>

<!-- End EncapsulatedPKIData -->

<!-- Start time-stamp container types -->

<!ELEMENT Include EMPTY>
<!ATTLIST Include
    URI CDATA #REQUIRED
    referencedData CDATA #IMPLIED
>

<!ELEMENT ReferenceInfo (DigestMethod, DigestValue)>
<!ATTLIST ReferenceInfo
    Id ID #IMPLIED
    URI CDATA #IMPLIED
>
<!ELEMENT XAdESTimeStamp (Include*, CanonicalizationMethod?,
(EncapsulatedTimeStamp | XMLTimeStamp)+)>
<!ATTLIST XAdESTimeStamp
    Id ID #IMPLIED
>

<!ELEMENT OtherTimeStamp (ReferenceInfo+, CanonicalizationMethod?,
(EncapsulatedTimeStamp | XMLTimeStamp))>
<!ATTLIST OtherTimeStamp
    Id ID #IMPLIED
>

<!ELEMENT EncapsulatedTimeStamp (#PCDATA)>
<!ATTLIST EncapsulatedTimeStamp
    Id ID #IMPLIED
>

<!ELEMENT XMLTimeStamp (#PCDATA  %XMLTimeStamp.ANY; )*>

<!-- End time-stamp tokens container  -->

<!-- Start container types -->
```

```
<!-- Start QualifyingProperties -->

<!ELEMENT QualifyingProperties (SignedProperties?, UnsignedProperties?)>
<!ATTLIST QualifyingProperties
    Target CDATA #REQUIRED
    Id ID #IMPLIED
>

<!ELEMENT SignedProperties (SignedSignatureProperties,
SignedDataObjectProperties?)>
<!ATTLIST SignedProperties
    Id ID #IMPLIED
>

<!ELEMENT UnsignedProperties (UnsignedSignatureProperties?,
UnsignedDataObjectProperties?)>
<!ATTLIST UnsignedProperties
    Id ID #IMPLIED
>

<!-- End QualifyingProperties -->

<!-- Start SignedSignatureProperties, SignedDataObjectProperties,
UnsignedSignatureProperties, UnsignedDataObjectProperties -->

<!ELEMENT SignedSignatureProperties (SigningTime?, SigningCertificate?,
SignaturePolicyIdentifier?, SignatureProductionPlace?, SignerRole?)>
<!ATTLIST SignedSignatureProperties
    Id ID #IMPLIED
>

<!ELEMENT SignedDataObjectProperties (DataObjectFormat*,
CommitmentTypeIndication*, AllDataObjectsTimeStamp*,
IndividualDataObjectsTimeStamp*)>
<!ATTLIST SignedDataObjectProperties
    Id ID #IMPLIED
>

<!ELEMENT UnsignedSignatureProperties (CounterSignature | SignatureTimeStamp
| CompleteCertificateRefs | CompleteRevocationRefs | AttributeCertificateRefs
 | AttributeRevocationRefs | (SigAndRefsTimeStamp | RefsOnlyTimeStamp) |
CertificateValues | RevocationValues | AttrAuthoritiesCertValues
| AttributeRevocationValues | %Any.ANY; | ArchiveTimeStamp)+>
<!ATTLIST UnsignedSignatureProperties
    Id ID #IMPLIED
>

<!ELEMENT UnsignedDataObjectProperties (UnsignedDataObjectProperty*)>
<!ATTLIST UnsignedDataObjectProperties
    Id ID #IMPLIED
>

<!ELEMENT UnsignedDataObjectProperty (#PCDATA   %Any.ANY;)*>

<!-- End SignedSignatureProperties, SignedDataObjectProperties,
UnsignedSignatureProperties, UnsignedDataObjectProperties -->

<!-- Start QualifyingPropertiesReference -->

<!ELEMENT QualifyingPropertiesReference EMPTY>
<!ATTLIST QualifyingPropertiesReference
    URI CDATA #REQUIRED
    Id ID #IMPLIED
>

<!-- End QualifyingPropertiesReference -->

<!-- End container types -->

<!-- Start SigningTime -->

<!ELEMENT SigningTime (#PCDATA)>

<!-- End SigningTime -->

<!-- Start SigningCertificate -->

<!ELEMENT SigningCertificate (Cert+)>
<!ELEMENT Cert (CertDigest, IssuerSerial)>
```

```
<!ATTLIST Cert
    URI CDATA #IMPLIED
>
<!ELEMENT CertDigest (DigestMethod, DigestValue)>
<!ELEMENT IssuerSerial (X509IssuerName, X509SerialNumber)>
<!ELEMENT X509IssuerName (#PCDATA)>
<!ELEMENT X509SerialNumber (#PCDATA)>

<!-- End SigningCertificate -->

<!-- Start SignaturePolicyIdentifier -->

<!ELEMENT SignaturePolicyIdentifier (SignaturePolicyId |
SignaturePolicyImplied)>
<!ELEMENT SignaturePolicyImplied ANY>
<!ELEMENT SignaturePolicyId (SigPolicyId, Transforms?, SigPolicyHash,
SigPolicyQualifiers?)>
<!ELEMENT SigPolicyId (Identifier, Description?, DocumentationReferences?)>
<!ELEMENT SigPolicyHash (DigestMethod, DigestValue)>
<!ELEMENT SigPolicyQualifiers (SigPolicyQualifier+)>
<!ELEMENT SigPolicyQualifier (#PCDATA %Any.ANY; )*>

<!-- End SignaturePolicyIdentifier -->

<!-- Start SPURI and SPUserNotice -->

<!ELEMENT SPURI (#PCDATA)>
<!ELEMENT SPUserNotice (NoticeRef?, ExplicitText?)>
<!ELEMENT NoticeRef (Organization, NoticeNumbers)>
<!ELEMENT ExplicitText (#PCDATA)>
<!ELEMENT Organization (#PCDATA)>
<!ELEMENT NoticeNumbers (int*)>
<!ELEMENT int (#PCDATA)>

<!-- End SPURI and SPUserNotice -->

<!-- Start CounterSignature -->

<!ELEMENT CounterSignature (Signature)>

<!-- End CounterSignature -->

<!-- Start DataObjectFormat -->

<!ELEMENT DataObjectFormat (Description?, ObjectIdentifier?, MimeType?,
Encoding?)>
<!ATTLIST DataObjectFormat
    ObjectReference CDATA #REQUIRED
>
<!ELEMENT MimeType (#PCDATA)>
<!ELEMENT Encoding (#PCDATA)>

<!-- End DataObjectFormat -->

<!-- Start CommitmentTypeIndication -->

<!ELEMENT CommitmentTypeIndication (CommitmentTypeId, (ObjectReference+ |
AllSignedDataObjects), CommitmentTypeQualifiers?)>
<!ELEMENT CommitmentTypeId (Identifier, Description?,
DocumentationReferences?)>
<!ELEMENT ObjectReference (#PCDATA)>
<!ELEMENT AllSignedDataObjects ANY>
<!ELEMENT CommitmentTypeQualifiers (CommitmentTypeQualifier*)>
<!ELEMENT CommitmentTypeQualifier (#PCDATA %Any.ANY; )*>

<!-- End CommitmentTypeIndication -->

<!-- Start SignatureProductionPlace -->

<!ELEMENT SignatureProductionPlace (City?, StateOrProvince?, PostalCode?,
CountryName?)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT StateOrProvince (#PCDATA)>
<!ELEMENT PostalCode (#PCDATA)>
<!ELEMENT CountryName (#PCDATA)>

<!-- End SignatureProductionPlace -->

<!-- Start SignerRole -->
```

```
<!ELEMENT SignerRole (ClaimedRoles?, CertifiedRoles?)>
<!ELEMENT ClaimedRoles (ClaimedRole+)>
<!ELEMENT CertifiedRoles (CertifiedRole+)>
<!ELEMENT ClaimedRole (#PCDATA %Any.ANY; )*>
<!ELEMENT CertifiedRole (#PCDATA)>
<!ATTLIST CertifiedRole
    Id ID #IMPLIED
    Encoding CDATA #IMPLIED
>

<!-- End SignerRole -->

<!-- Start AllDataObjectsTimeStamp, IndividualDataObjectsTimeStamp,
SignatureTimeStamp -->

<!ELEMENT AllDataObjectsTimeStamp (CanonicalizationMethod?,
(EncapsulatedTimeStamp | XMLTimeStamp)+)>
<!ATTLIST AllDataObjectsTimeStamp
    Id ID #IMPLIED
>

<!ELEMENT IndividualDataObjectsTimeStamp (Include+, CanonicalizationMethod?,
(EncapsulatedTimeStamp | XMLTimeStamp)+)>
<!ATTLIST IndividualDataObjectsTimeStamp
    Id ID #IMPLIED
>

<!ELEMENT SignatureTimeStamp (CanonicalizationMethod?, (EncapsulatedTimeStamp
| XMLTimeStamp)+)>
<!ATTLIST SignatureTimeStamp
    Id ID #IMPLIED
>

<!-- End AllDataObjectsTimeStamp, IndividualDataObjectsTimeStamp,
SignatureTimeStamp -->

<!-- Start CompleteCertificateRefs -->

<!ELEMENT CompleteCertificateRefs (CertRefs)>
<!ATTLIST CompleteCertificateRefs
    Id ID #IMPLIED
>
<!ELEMENT CertRefs (Cert+)>

<!-- End CompleteCertificateRefs -->

<!-- Start AttributeCertificateRefs -->

<!ELEMENT AttributeCertificateRefs (CertRefs)>
<!ATTLIST AttributeCertificateRefs
    Id ID #IMPLIED
>

<!-- End AttributeCertificateRefs -->

<!-- Start CompleteRevocationRefs -->

<!ELEMENT CompleteRevocationRefs (CRLRefs?, OCSPRefs?, OtherRefs?)>
<!ATTLIST CompleteRevocationRefs
    Id ID #IMPLIED
>
<!ELEMENT CRLRefs (CRLRef+)>
<!ELEMENT OCSPRefs (OCSPRef+)>
<!ELEMENT OtherRefs (OtherRef+)>

<!ELEMENT CRLRef (DigestAlgAndValue, CRLIdentifier?)>
<!ELEMENT OCSPRef (OCSPIdentifier, DigestAlgAndValue?)>
<!ELEMENT OtherRef (#PCDATA %Any.ANY; )*>

<!ELEMENT DigestAlgAndValue (DigestMethod, DigestValue)>
<!ELEMENT CRLIdentifier (Issuer, IssueTime, Number?)>
<!ATTLIST CRLIdentifier
    URI CDATA #IMPLIED
>
<!ELEMENT OCSPIdentifier (ResponderID, ProducedAt)>
<!ATTLIST OCSPIdentifier
    URI CDATA #IMPLIED
>
```

```
<!ELEMENT Issuer (#PCDATA)>
<!ELEMENT IssueTime (#PCDATA)>
<!ELEMENT Number (#PCDATA)>

<!ELEMENT ResponderID (ByName | ByKey)>
<!ELEMENT ByName  (#PCDATA)>
<!ELEMENT ByKey  (#PCDATA)>
<!ELEMENT ProducedAt (#PCDATA)>

<!-- End CompleteRevocationRefs -->

<!-- Start AttributeRevocationRefs -->

<!ELEMENT AttributeRevocationRefs (CRLRefs?, OCSPRefs?, OtherRefs?)>
<!ATTLIST AttributeRevocationRefs
    Id ID #IMPLIED
>

<!-- End AttributeRevocationRefs -->

<!-- Start SigAndRefsTimeStamp, RefsOnlyTimeStamp  -->

<!ELEMENT SigAndRefsTimeStamp (Include*, CanonicalizationMethod?,
(EncapsulatedTimeStamp | XMLTimeStamp)+)>
<!ATTLIST SigAndRefsTimeStamp
    Id ID #IMPLIED
>

<!ELEMENT RefsOnlyTimeStamp (Include*, CanonicalizationMethod?,
(EncapsulatedTimeStamp | XMLTimeStamp)+)>
<!ATTLIST RefsOnlyTimeStamp
    Id ID #IMPLIED
>

<!-- End SigAndRefsTimeStamp, RefsOnlyTimeStamp  -->

<!-- Start CertificateValues -->

<!ELEMENT CertificateValues (EncapsulatedX509Certificate |
OtherCertificate)*>
<!ATTLIST CertificateValues
    Id ID #IMPLIED
>

<!ELEMENT EncapsulatedX509Certificate (#PCDATA)>
<!ATTLIST EncapsulatedX509Certificate
    Id ID #IMPLIED
    Encoding CDATA #IMPLIED
>
<!ELEMENT OtherCertificate (#PCDATA %Any.ANY;)*>

<!ELEMENT AttrAuthoritiesCertValues (EncapsulatedX509Certificate |
OtherCertificate)*>
<!ATTLIST CertificateValues
    Id ID #IMPLIED
>

<!-- Start RevocationValues -->

<!ELEMENT RevocationValues (CRLValues?, OCSPValues?, OtherValues?)>
<!ATTLIST RevocationValues
    Id ID #IMPLIED
>

<!ELEMENT CRLValues (EncapsulatedCRLValue+)>
<!ELEMENT OCSPValues (EncapsulatedOCSPValue+)>
<!ELEMENT OtherValues (OtherValue+)>

<!ELEMENT EncapsulatedCRLValue (#PCDATA)>
<!ATTLIST EncapsulatedCRLValue
    Id ID #IMPLIED
    Encoding CDATA #IMPLIED
>
<!ELEMENT EncapsulatedOCSPValue (#PCDATA)>
<!ATTLIST EncapsulatedOCSPValue
    Id ID #IMPLIED
    Encoding CDATA #IMPLIED
>
```

```
<!ELEMENT OtherValue (#PCDATA %Any.ANY;  )*>

<!-- End RevocationValues -->

<!ELEMENT AttributeRevocationValues (CRLValues?, OCSPValues?, OtherValues?)>
<!ATTLIST RevocationValues
    Id ID #IMPLIED
>
<!-- Start ArchiveTimeStamp -->

<!ELEMENT ArchiveTimeStamp (Include*, CanonicalizationMethod?,
(EncapsulatedTimeStamp | XMLTimeStamp)+)>
<!ATTLIST ArchiveTimeStamp
    Id ID #IMPLIED
>


<!-- End ArchiveTimeStamp -->
```

# Annex F (informative): Incorporation of qualifying properties

As stated in the normative part of the present document, new elements have been defined to incorporate properties (both signed and unsigned) that qualify the whole signature, the signer or individual signed data objects: `QualifyingProperties`, `SignedProperties`, `UnsignedProperties`, `SignedSignatureProperties`, `UnsignedSignatureProperties`, `SignedDataObjectProperties` and `UnsignedDataProperties`.

Annex E shows an example of direct incorporation of qualifying properties and one example of indirect incorporation of these properties.

Below follows the resulting general structure of direct incorporation.

```
<ds:Signature ID?>

  <ds:SignedInfo>
    <ds:CanonicalizationMethod/>
    <ds:SignatureMethod/>
    (<ds:Reference URI? >
      (<ds:Transforms>)?
      <ds:DigestMethod/>
      <ds:DigestValue/>
    </Reference>)+
  </ds:SignedInfo>

  <ds:SignatureValue/>

  (<ds:KeyInfo>)?

  <ds:Object>

    <QualifyingProperties>

      <SignedProperties>

        <SignedSignatureProperties>
          <!-- Collection of signed XML elements with
          properties qualifying the signature or the
          signer -->
        </SignedSignatureProperties>

        <SignedDataObjectProperties>
          <!-- Collection of signed XML elements with
          properties individually qualifying signed data
          objects -->
      </SignedDataObjectPropertiesSigned>

      </SignedProperties>

      <UnsignedProperties>

        </UnsignedSignatureProperties>
          <!-- Collection of unsigned XML elements with
          properties qualifying signature or signer -->
        </UnsignedSignatureProperties>

        <UnSignedDataObjectProperties>
          <!-- Collection of unsigned XML elements with
          properties individually qualifying signed
          data objects -->
        </UnSignedDataObjectProperties>

      </UnsignedProperties>

    </QualifyingProperties>

  </ds:Object>

</ds:Signature>
```

Below follows an example showing the inclusion of three sets of qualifying properties:

- The first one includes signed properties qualifying the signature, namely:
    - the time of signature production (element `SigningTime`);
    - a restricted set of references to certificates to be used in verifying a signature. This also includes a reference to the certificate containing the public key corresponding to the private key used in the signature computation (element `SigningCertificate`);
    - an identification of the signature policy under which the signature has been produced and will have to be verified (element `SignaturePolicyIdentifier`).
- The second one includes signed properties qualifying the signed data object, namely:
    - a time-stamp of the signed data object, proving that the content has been produced before the time indicated in the time-stamp (element `AllDataObjectsTimeStamp`);
    - an indication of the format of the signed object (element `DataObjectFormat`).
- The third one includes unsigned properties qualifying the signature, namely:
    - a time-stamp on the electronic signature itself, proving that the signature was produced before the time indicated by such time-stamp (element `SignatureTimeStamp`);
    - the references to the full set of CA certificates that the verifier of the electronic signature has used to validate the electronic signature (element `CompleteCertificateRefs`);
    - the references to the revocation material (CRLs or OCSP responses) used in the validation of the signer and CA certificates used the full to validate the electronic signature (element `CompleteRevocationRefs`);
    - the time-stamp generated over the electronic signature with the aforementioned qualifying information (element `SigAndRefsTimeStamp`);
    - the full set of CA certificates that the verifier of the electronic signature has used to validate the electronic signature (element `CertificateValues`);
    - the revocation material (CRLs or OCSP responses) used in the validation of the signer and CA certificates used the full to validate the electronic signature (element `RevocationValues`).

```
[s01]<ds:Signature Id="SignatureId" xmlns:ds="http://www.w3.org/2000/09/xmldsig#>
[s02]  <ds:SignedInfo Id="SignedInfoId">
[s03]    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
[s04]    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
[s05]    <ds:Reference URI="http://www.etsi.org/docToBeSigned" Id="FirstSignedDocument">
[s06]      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[s07]      <ds:DigestValue>… … …</ds:DigestValue>
[s08]    </ds:Reference>
[s09]    <ds:Reference URI="#SignedPropertiesId"
Type="http://uri.etsi.org/01903#SignedProperties">
[s10]      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[s11]      <ds:DigestValue>… … …</ds:DigestValue>
[s12]    </ds:Reference>
[s13]  </ds:SignedInfo>
[s14]  <ds:SignatureValue Id="SignatureValueId">… … … </ds:SignatureValue>
[s15]  <ds:KeyInfo>… … …</ds:KeyInfo>
[s16]  <ds:Object >

[s17]     <QualifyingProperties  Id="QualifyingPropertiesId" Target="#SignatureId"
xmlns="http://uri.etsi.org/01903/v1.3.2#" xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

[s18]      <SignedProperties Id="SignedPropertiesId">
[s19]        <SignedSignatureProperties >
[s20]          <SigningTime>… … …</SigningTime>
[s21]          <SigningCertificate>… … …</SigningCertificate >
[s22]          <SignaturePolicyIdentifier>… … …</ SignaturePolicyIdentifier >
[s23]        </SignedSignatureProperties>
[s24]        <SignedDataObjectProperties>
[s25]          <DataObjectFormat>… … …</DataObjectFormat>
```

```
[s26]          <AllDataObjectsTimeStamp Id="AllDataObjectsTimeStampId">… … …
</AllDataObjectsTimeStamp>
[s27]        </SignedDataObjectProperties>
[s28]      </SignedProperties>
[s29]      <UnsignedProperties >
[s30]        <UnsignedSignatureProperties>
[s31]          <SignatureTimeStamp
Id="SignatureTimeStampId">…  …  …</SignatureTimeStamp>
[s32]          <CompleteCertificateRefs Id="CompleteCertificateRefsId">…  …
…</CompleteCertificateRefs>
[s33]          <CompleteRevocationRefs
Id="CompleteRevocationRefsId">…  …  …</CompleteRevocationRefs>
[s34]          <SigAndRefsTimeStamp
Id="SigAndRefsTimeStampId">… … …</SigAndRefsTimeStamp>
[s35]          <CertificateValues
Id="CertificateValuesId">…  …  …  …</CertificateValues>
[s36]          <RevocationValues
Id="RevocationValuesId">…  …  …</RevocationValues>
[s37]        </UnsignedSignatureProperties>
[s38]      </UnsignedProperties>
[s39]    </QualifyingProperties>
[s40]  </ds:Object>
[s41]</ds:Signature>
```

| | |
|---|---|
| [s01] | Beginning of the XML signature. The namespace by default is the namespace defined in XMLDSIG. |
| [s02]-[s13] | The `ds:SignedInfo` element contains the information that is actually signed. |
| [s03] | The `ds:CanonicalizationMethod` element indicates the algorithm used to get a canonical representation of the `ds:SignedInfo` element before being signed. |
| [s04] | The `ds:SignatureMethod` indicates the algorithms used to sign `ds:SignedInfo`. |
| [s05] to [s16] | `ds:Reference` elements contain the digest value and indication on the digest algorithm for each data object that has to be (indirectly) signed. Each one also has a reference to the corresponding data object. These elements also have the `Id` attribute that can be used to make individual references each one of them. |
| [s05-s08] | The first `ds:Reference` element. Its URI attribute references the data object that has to be signed. `ds:DigestMethod` indicates the digest algorithm (sha1 in this case) and `ds:DigestValue` contains the base64 encoded digest value. |
| [s09-s12] | The second `ds:Reference` element. Its URI attribute points to the `SignedProperties` element (using the URI attribute) that contains the whole set of signed properties. `ds:DigestMethod` indicates the digest algorithm (sha1 in this case) and `ds:DigestValue` contains the digest value filtered in base. This means that the digest value of that `SignedProperties` is included in `ds:SignedInfo` and in consequence signed when this element is signed. The `Type` attribute indicates that this element is a reference to the `SignatureProperties` element as mandated in clause 6.3.1. |
| [s14] | `ds:SignatureValue` contains the computed digital signature of `ds:SignedInfo` in base64. |
| [s15] | `ds:KeyInfo` contains cryptographic material to verify the signature. |
| [s16-s40] | `ds:Object` contains three elements with the properties qualifying both the signature and the signed data object. |
| [s17-39] | `QualifyingProperties` contains the full set of qualifying properties both signed (`SignedProperties`) and unsigned (Uns`ignedProperties`). The namespace by default is changed for this element and its contents to the one defined as namespace by default in the schema definition given in the present document in order not to have to qualify the whole set of elements. Additionally, as elements already defined in [3] are used in the definitions, its namespace is also defined (prefix `ds`). |

| | |
|---|---|
| [s18-s28] | `SignedProperties` contains the whole set of qualifying properties that are signed grouped in two sequences. The first one (`SignedSignatureProperties`) contains all the signed properties that qualify the signature. The second one (`SignedDataObjectProperties`) contains all the signed properties that individually qualify each signed data object. |
| [s19-ss23] | `SignedSignatureProperties` contains all the signed properties that qualify the signature (`SigningTime`, `SigningCertificate`, `SignaturePolicyIdentifier`). |
| [s20] | `signingTime` contains the value of the signing instant when the signature has been computed. |
| [s21] | `SigningCertificate` contains, as stated above, a restricted set of references to certificates to be used in verifying a signature. |
| [s24-27] | `SignedDataObjectProperties` contains all the signed properties that individually qualify each signed data object (`AllDataObjectsTimeStamp, DataObjectFormat`). |
| [s25] | `DataObjectFormat` identifies the format of the signed data object. |
| [s26] | `AllDataObjectsTimeStamp` is a time-stamp issued for the signed data object. |
| [s29-38] | `UnsignedProperties` contains the whole set of qualifying properties that are NOT signed. |
| [s30-s37] | `UnsignedSignatureProperties` the whole set of unsigned properties that qualify the signature. |
| [s31] | `SignatureTimeStamp` contains a time-stamp for the signature itself. |
| [s32] | `CompleteCertificateRefs` contains references to CA certificates in the certification path used to verify the signature. |
| [s33] | `CompleteRevocationRefs` contains references to revocation information used to verify the signature. |
| [s34] | `SigAndRefsTimeStamp` contains a time-stamp over the XAdES-C form of the electronic signature. |
| [s35] | `CertificateValues` contains the values of the certificates referenced in `CompleteCertificateRefs`. |
| [s36] | `RevocationValues` contains the revocation data used to validate the electronic signature. |

NOTE: The tree shown in the example above does not explicitly show certain optional XML elements (like `ds:Transforms` For a complete description of this tree see W3C/IETF Recommendation "XML-Signature Core Syntax and Processing" [3].

Below follows the example of indirect incorporation of all the unsigned properties. In this example, the signed properties will be directly incorporated into the `ds:Signature` element as in the previous example. However, the unsigned properties will be separately stored in other place. To incorporate these properties use is made of the `QualifyingPropertiesReference` element pointing to the element containing them.

Below follows the contents of the XAdES itself that could be located, for instance, at the URI http://uri.etsi.org/01903/v1.3.2/Indirect-Incorporation/Signature2:

```
[s01]<ds:Signature Id="Signature2Id" xmlns:ds="http://www.w3.org/2000/09/xmldsig#>
[s02]  <ds:SignedInfo Id="SignedInfoId">
 [s03]    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
[s04]    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
[s05]    <ds:Reference URI="http://www.etsi.org/docToBeSigned" Id="FirstSignedDocument">
[s06]      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[s07]      <ds:DigestValue>… … …</ds:DigestValue>
[s08]    </ds:Reference>
[s09]    <ds:Reference URI="#SignedPropertiesId"
Type=http://uri.etsi.org/01903#SignedProperties>
[s10]      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[s11]      <ds:DigestValue>… … …</ds:DigestValue>
[s12]    </ds:Reference>
[s13]  </ds:SignedInfo>
[s14]  <ds:SignatureValue Id="SignatureValueId">… … … </ds:SignatureValue>
```

```
[s15]  <ds:KeyInfo>… … …</ds:KeyInfo>

[s16]  <ds:Object >
[s17]     <QualifyingProperties  Id="QualifyingProperties" Target="#SignatureId"
xmlns="http://uri.etsi.org/01903/v1.3.2#" xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
[s18]      <SignedProperties Id="SignedPropertiesId">
[s19]        <SignedSignatureProperties >
[s20]          <SigningTime>… … …</SigningTime>
[s21]          <SigningCertificate>… … …</SigningCertificate >
[s22]          <SignaturePolicyIdentifier>… … …</SignaturePolicyIdentifier >
[s23]        </SignedSignatureProperties>
[s24]        <SignedDataObjectProperties>
[s25]          <DataObjectFormat>… … …</DataObjectFormat>
[s26]          <AllDataObjectsTimeStamp Id="AllDataObjectsTimeStampId">… … …
</AllDataObjectsTimeStamp>
[s27]        </SignedDataObjectProperties>
[s28]      </SignedProperties>
[s29]    </QualifyingProperties>

[s30]    <QualifyingPropertiesReference
URI="http://uri.etsi.org/01903/v1.3.2/Indirect-Incorporation/example1#QualifyingPropertiesId">
[s31]    </QualifyingPropertiesReference>
[s32]  </ds:Object>
[s33]</ds:Signature>
```

| | |
|---|---|
| [s1-s29] | These lines are the same as in the first example. They show how the signed properties are directly incorporated. |
| [s30-s32] | These lines show how to indirectly incorporate the unsigned properties stored in other place using the `QualifyingPropertiesReference` element. |
| [s30] | The `URI` attribute contains the URI pointing to the `QualifyingProperties` element that contains those qualifying properties that are being indirectly incorporated. In this case, it points to a file that could be found in http://uri.etsi.org/01903/v1.3.2/Indirect-Incorporation/eample1, which contains this element. |

This example ends showing that part of the file that could be found in http://uri.etsi.org/01903/v1.3.2/Indirect-Incorporation/example1 that contains the `QualifyingProperties` element referenced in the `QualifyingPropertiesReference`.

```
<!-- This is the part of the file found in
http://uri.etsi.org/01903/v1.3.2/Indirect-Incorporation/Signature2 that contains the
QualifyingProperties element containing the unsigned properties that are indirectly incorporated in
the advanced electronic signature -->

[si]     <QualifyingProperties Id="QualifyingPropertiesId" Target="http://uri.etsi.org/01903/v1.3.2/
Indirect-Incorporation/Signature2#Signature2Id" xmlns="http://uri.etsi.org/01903/v1.3.2#"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
[si+1]      <UnsignedProperties >
[si+2]        <UnsignedSignatureProperties>
[si+3]          <SignatureTimeStamp
Id="SignatureTimeStampId">…  …  …</SignatureTimeStamp>
[si+4]          <CompleteCertificateRefs Id="CompleteCertificateRefsId">…  …
…</CompleteCertificateRefs>
[si+5]          <CompleteRevocationRefs
Id="CompleteRevocationRefsId">…  …  …</CompleteRevocationRefs>
[si+6]          <SigAndRefsTimeStamp
Id="SigAndRefsTimeStampId">… … …</SigAndRefsTimeStamp>
[si+7]          <CertificateValues
Id="CertificateValuesId">…  …  …</CertificateValues>
[si+8]          <RevocationValues
Id="RevocationValuesId">…  …  …</RevocationValues>
[si+9]        </UnsignedSignatureProperties>
[si+10]     </UnsignedProperties>
[si+11]  </QualifyingProperties>

<!-- Below would follow the rest of the file -->
```

In the example above the `QualifyingProperties` element is shown that is part of the file that could be found in http://uri.etsi.org/01903/v1.3.2/Indirect-Incorporation/example1 and that is pointed by the `URI` element in the `QualifyingPropertiesReference` in the advanced electronic signature.

# Annex G (informative): Details on XAdES signatures validation

This annex does not aim at specifying normative procedures for validating XAdES signatures, as the eventual acceptance or rejection of a specific electronic signature depends on both technical and non-technical considerations, these last ones being highly dependent on specific contexts. Instead, this annex gives only informative details on technical steps, which are likely to be common to most of the contexts where electronic signatures are used, and that verifiers SHOULD perform during the validation of such signatures.

The present annex is structured in two clauses.

The first one presents an example of how, after receiving a XAdES-EPES signature, a verifier could make to evolve it, as the validation process advances, towards more advanced XAdES forms, namely, XAdES-T, XAdES-C, XAdES-X, XAdES-X-L and XAdES-A. Other evolutions in the received signatures are, of course, possible, and it is expected to be dependant on the rules applying in the specific contexts.

The second clause enumerates technical rules for verification of the different qualifying properties present in XAdES that verifiers SHOULD follow while verifying XAdES electronic signatures.

## G.1 Signatures evolution example

The present specification does not preclude the signature generator to collect validation data and generate up to a XAdES-C form if she may wait the time required for collecting such validation data and generate the corresponding XAdES unsigned properties before sending the signature to its destine. Nevertheless, contexts where signers do not add such validation data are more likely to occur.

This clause gives an overview of how a recipient of a XAdES-EPES signature could make to evolve it, as the validation process advances, towards more advanced XAdES forms.

Figure G.1 shows the contents of the XAdES-EPES signature arrived to the verifier with an indication of the incorporated signed properties.

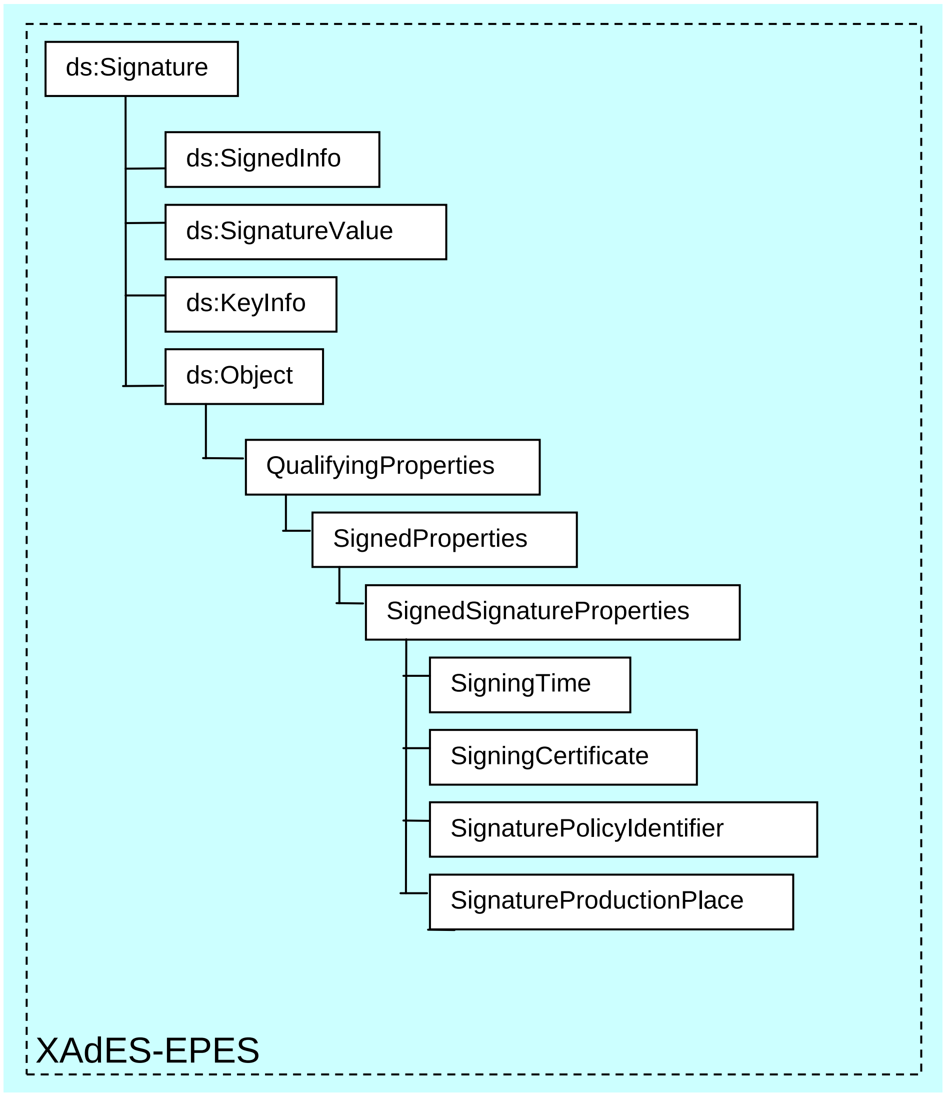


**Figure G.1: Initial XAdES-EPES signature**

Note that this is a XAdES-EPES signature built on a XAdES-BES because it contains `SigningCertificate` and `SignaturePolicyIdentifier` properties.

As said before the verifier may collect all the additional data that forms the electronic signature. Figures below and subsequent description, show how she may build up a complete electronic signature as she goes through the verification process.

Soon after receiving the XAdES-EPES signature, the verifier may check the signature value (1). Afterwards the validation process shall at least add one time-stamp on the signature (2), generating a XAdES-T form. The validation process may also validate the electronic signature, using additional data (e.g. certificates, CRL, etc.) provided by trusted service providers, and check whether the signature meets the requirements specified in the signature policy indicated by the `SignaturePolicyIdentifier` property. If the validation process result is validation incomplete, then the output from this stage is the ES-T. This process is shown in figure G.2.

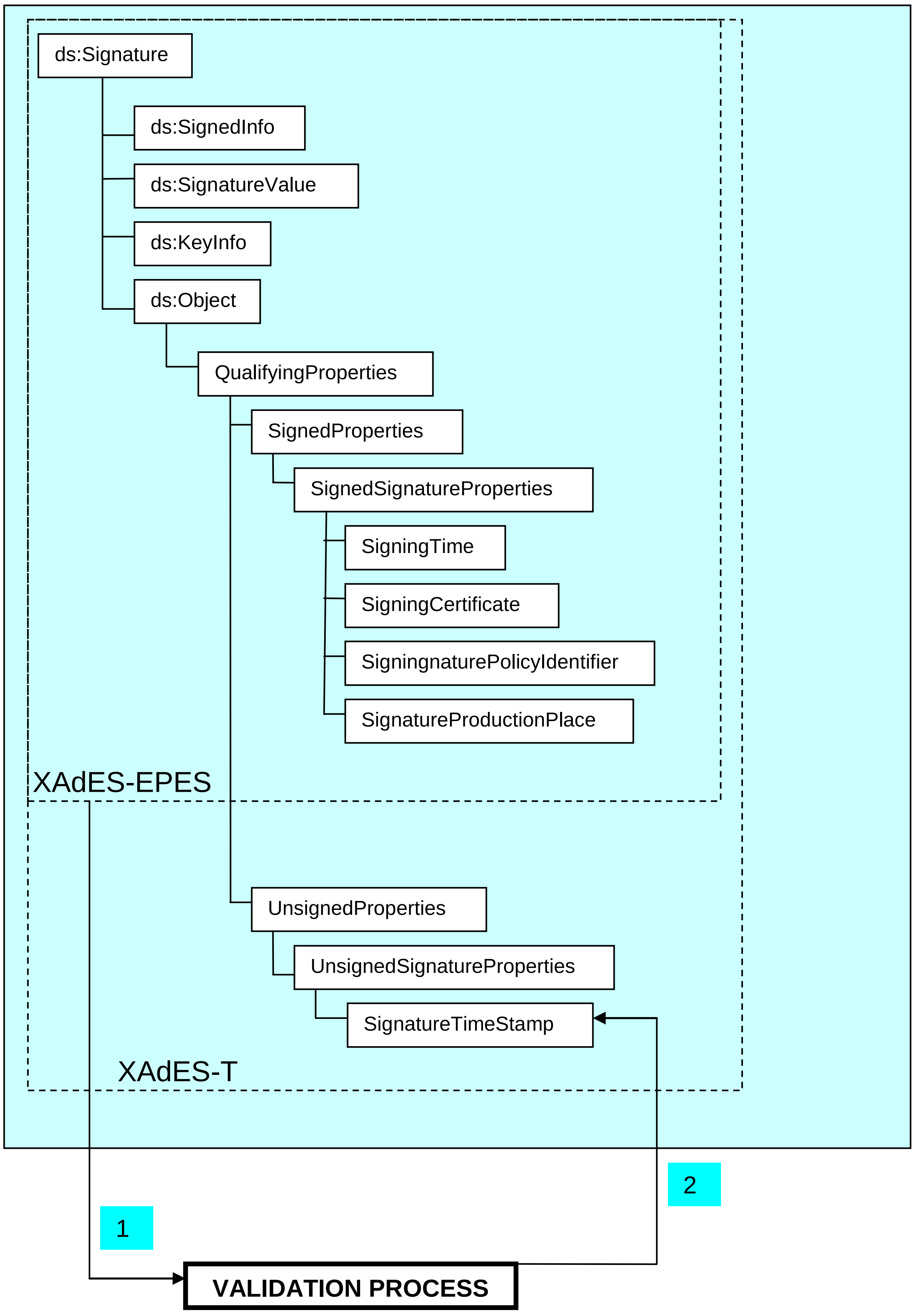


**Figure G.2: Generation of XAdES-T**

## G.1.1 Example of path to archival form with validation data references

To ascertain the validity status as Valid or Invalid and communicate that to the user, all the additional validation data must be available, including the complete certificate and revocation information.

Once the verifier has got such information, she may complete the validation checks and generate the XAdES-C form (3), adding `CompleteCertificateRefs` and `CompleteRevocationRefs` properties, as shown in figure G.3.

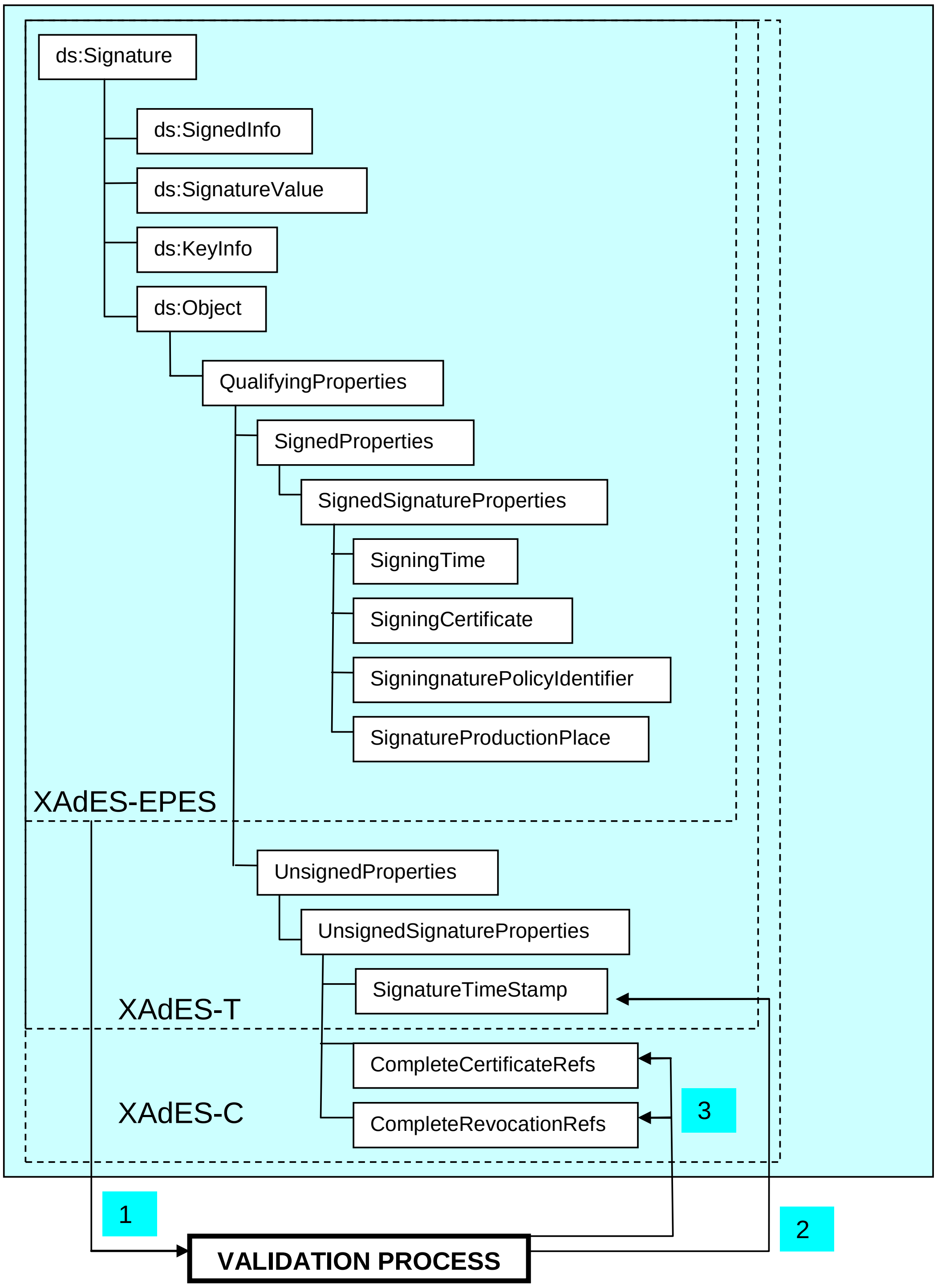


**Figure G.3: Generation of XAdES-C**

When the validation process creates the ES-C it may also create extended forms of validation data, namely XAdES-X, in its two variants: the one including `SigAndRefsTimeStamp` or the one including `RefsOnlyTimeStamp` properties (4) as shown in figure G.4.

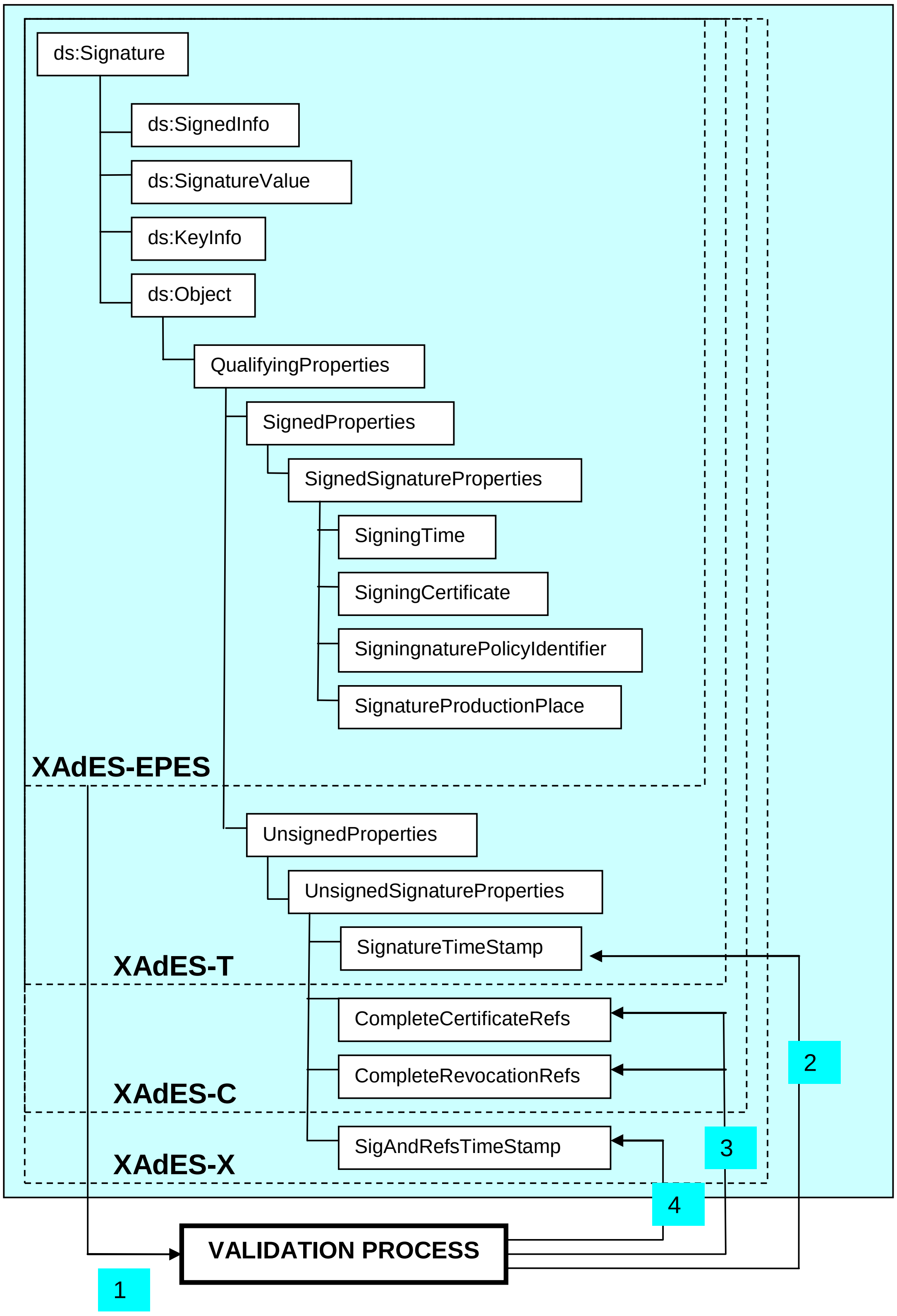


**Figure G.4: Generation of XAdES-X**

Before the algorithms used in any of electronic signature become or are likely, to be compromised or rendered vulnerable in the future, it may be necessary to add all the values of the validation and user data and time-stamp the whole signature, obtaining a XAdES-A form as shown in figure G.5.

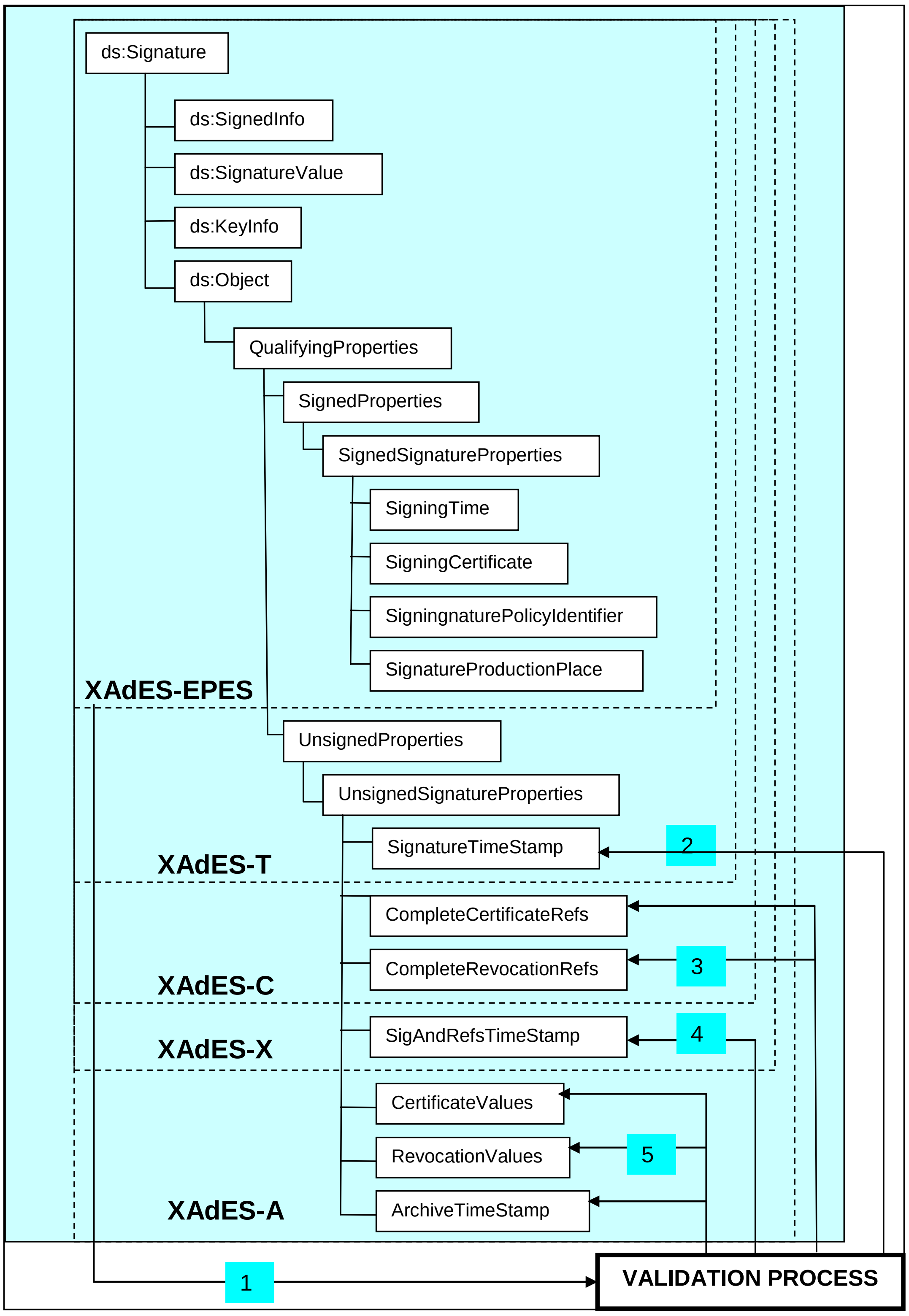


**Figure G.5: Generation of XAdES-A**

### G.1.2 Example of path to archival form without validation data references

Alternatively, the verifier could also generate an archival form without incorporating any reference to validation data, with the addition of the validation values and the `ArchiveTimeStamp` property, as indicated in figure G.6.

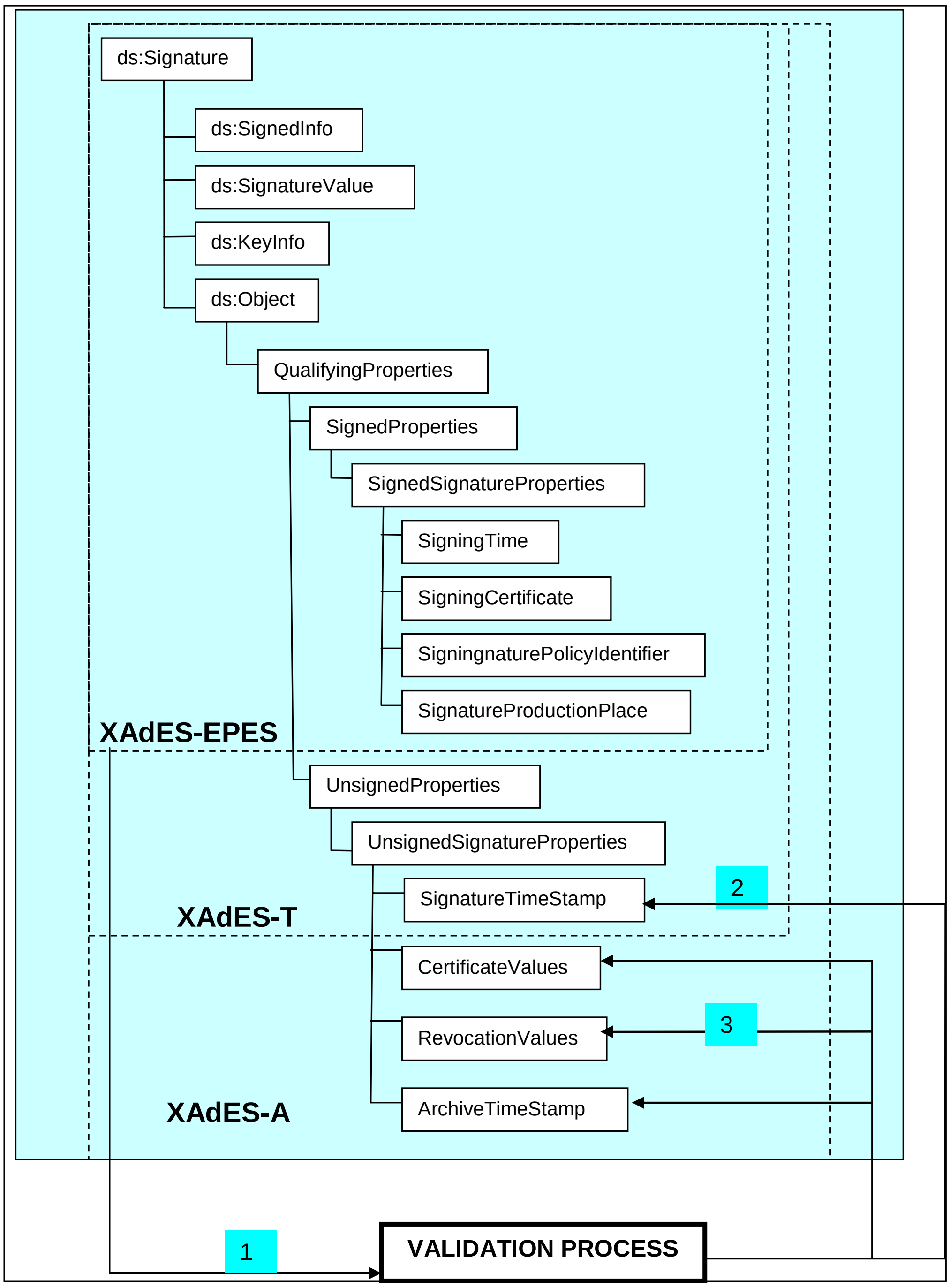


**Figure G6**

## G.2 Verification technical rules

This clause presents a set of technical rules that verifiers of XAdES signatures should follow.

First a short description of the relationship of this annex with already existing standards that include verification procedures and rules is given.

Then the technical rules for verification of the XAdES signatures are presented.

### G.2.1 Relationship with other standard verification procedures

XAdES signatures build on XMLDSIG signatures. In consequence, verification rules described in XMLDSIG [3] also apply to XAdES signatures.

As for the process of verification of validity of the certification path, the usual verification procedures already standardized apply, like the one in RFC 3280 [14] clause 6 and the rules established in RFC 2560 [9] for OCSP protocol.

### G.2.2 Verification procedure

This clause provides technical rules that verifiers should follow in addition to the aforementioned ones, while verifying the qualifying properties present in XAdES signatures.

#### G.2.2.1 General Checks

The verification process should assess whether the signature is a XAdES signature and if so, identify the specific form, by inspecting the different qualifying properties present. The verification process should not accept as XAdES signature any combination not aligned with those established in the normative part of the present document or the extended forms defined in the informative annex B.

As part of the aforementioned process, the verifier should check whether the `ds:SigningCertificate` property is present. If not, she should check that the `ds:KeyInfo` contains the signing certificate and is referenced by one of the `ds:Reference` elements in the signature, so that it is protected by the signature value.

During the aforementioned process, the verifier should also assess whether the incorporation of properties is direct (all the properties within one `ds:Object` element enveloped by the `ds:Signature` element) or indirect (denoted by the presence of `QualifyingPropertiesReference` elements within one of the `ds:Object` elements enveloped by the `ds:Signature`), and once this is done, check the correctness of incorporation of properties according to the rules stated in clause 6.3 of the normative part.

The verification process should also check the presence and the value ("`http://uri.etsi.org/01903#SignedProperties`") of the `Type` attribute in the `ds:Reference` element referencing the `ds:Object`, which encloses all the XAdES signed properties in the direct incorporation case, or the `QualifyingPropertiesReference` elements in the indirect incorporation case. If no Type attribute is present with such a value, the verification process should not accept the signature as a XAdES signature.

#### G.2.2.2 Getting certificates for verification

The normative part of the present specification states that the `CertificateValues` element "*contains the full set of certificates that have been used to validate the electronic signature, including the signer's certificate*" except those ones that are already present within the `ds:KeyInfo` element. In consequence, if the XAdES signature contains the `CertificateValues` property, then the verifier should use this property and the `ds:KeyInfo` for getting all the certificates required for performing the verification. The verifier should also check that the contents of these two elements actually form a valid certification path. If not, the verifier should assume that the verification process has failed.

If `CertificateValues` is not present but `CompleteCertificateRefs` is present, the verifier should get the certificates referenced there and check if they actually form a valid certification path. If not, the verifier should assume that the verification process has failed.

If neither CertificateValues nor CompleteCertificateRefs are present, the specific means by which the verifier can get the certification path are out of scope of the present specification.

The same rules apply for the `AttrAuthoritiesCertValues` property regarding to the retrieval of Attribute Authorities certificates and its usage in the validation of the attribute certificate(s) present in the `SignerRole` property.

### G.2.2.3 Getting certificates" status information for verification

The normative part of the present specification states that the `RevocationValues` element contains information on the status of all certificates required for verifying the electronic signature. At the time of writing of this clause, CRLs and OCSP responses are the only standardized contents of this property.

If the XAdES signature contains the `RevocationValues` property, then the verifier should use this property for getting the information on the status of all the aforementioned certificates. The verifier should also check if they actually provide adequate revocation information for all the certificates required for verifying the electronic signature. If not, the verifier should assume that the verification process has failed.

If `RevocationValues` is not present but `CompleteRevocationRefs` is present, the verifier should get the certificate status information data referenced there and check if they actually provide adequate revocation information for all the certificates required for verifying the electronic signature. If not, the verifier should assume that the verification process has failed.

If neither `RevocationValues` nor `CompleteRevocationRefs` are present, the specific means by which the verifier can get this information is out of the scope of the present specification.

The same rules apply for the `AttributeRevocationValues` property regarding to the retrieval of the status of the attribute certificate(s) present in the `SignerRole` property and the Attribute Authorities certificates and its usage in its validation.

### G.2.2.4 Checking `SigningTime`

The verification process should assess that the claimed time is previous to the time the signatures is being verified.

### G.2.2.5 Checking `SigningCertificate`

If the `CertificateValues` is present, the verifier could get it from this property or from within the `ds:KeyInfo` element. If the `CertificateValues` element is not present, the verifier may gain access to the signer"s certificate from within the `ds:KeyInfo,` if present, or by other means that are out of the scope of the present specification. In addition, the means allowing the verifier to identify the signer"s certificate are out of scope of the present document.

Once the verifier has gotten the signing certificate, she should check it against the references present in the `ds:SigningCertificate` property, if present. For doing this, and for each reference present in the property, the verifier should perform the following tasks:

1) Compare the name of the issuer and the serial number of the certificate with those indicated in the `IssuerSerial` element. For doing this, the application must follow the indications given in XMLDSIG clause 4.4.4 on how to generate the string corresponding to the issuer"s distinguished name. If not, take the next reference and re-start again in 1. If they match, continue with 2.

2) If the `ds:KeyInfo` contains the `ds:X509IssuerSerial` element, check that the issuer and the serial number indicated in both, that one and `IssuerSerial` from `SigningCertificate`, are the same.

3) Check that the content of `ds:DigestValue` is the result of digesting the certificate with the algorithm indicated in `ds:DigestMethod` and base64 encoding this digest.

If the verifier does not find any reference matching the signing certificate, the validation of this property should be taken as failed.

If `SigningCertificate` contains references to other certificates in the path, the verifier should proceed to check each of the certificates in the certification path against them.

Should this property contain one or more references to certificates other than those present in the certification path, the verifier should assume that a failure has occurred during the verification.

Should one or more certificates in the certification path not be referenced by this property, the verifier should assume that the verification is successful unless the signature policy mandates that references to all the certificates in the certification path must be present.

## G.2.2.6 Checking `SignaturePolicyIdentifier`

If this property is present and it is not implied, the verifier should:

1) Retrieve the electronic document containing the details of the policy, and identified by the contents of `SigPolicyId` element. Apply the transformations indicated in the `ds:Transforms` element of `SignaturePolicyId`, compute the digest of the resulting document using the algorithm indicated in `ds:DigestMethod` and check its value with the digest value present in `ds:DigestMethod`.

2) Should the `SignaturePolicyIdentifier` element have qualifiers, the verifier should manage them according to the rules that are stated by the policy applying within the specific scenario.

3) If the checks described before end successfully, the verifier should proceed to perform the checks mandated by the specific signature policy. The way used by the signature policy for presenting them and their description are out of the scope. TR 102 038 [12] specifies a "XML format for signature policies" that may be automatically processed.

If the signature policy is implied, the verifier should perform the checks mandated by the implicit signature policy.

## G.2.2.7 Checking Countersignatures

As stated in the normative part, countersignatures are managed in two ways:

1) XAdES signature contains a countersignature by using the `CounterSignature` property. When using direct incorporation, this property is an enveloped countersignature. The content of such property may also be a XAdES signature.

2) XAdES signature being a countersignature of one or more signatures. This is achieved by forcing the value "http://uri.etsi.org/01903#CounterSignedSignature" for the `Type` attribute of one or several `ds:Reference` elements.

If the `CounterSignature` property is present, the verifier should:

1) Check that the enclosed signature correctly references the `ds:SignatureValue` present in the countersigned XAdES signature.

2) Proceed to start a new verification process for the countersignature itself in the same way as for regular signatures. Should the content of `CounterSignature` be a XAdES signature, the technical checks described in the current annex should also be applied.

If a XAdES signature contains one or more `ds:Reference` elements with `Type` attributes whose values are "http://uri.etsi.org/01903#CounterSignedSignature", the verifier should take into account that this is a countersignature of another signature.

The specific rules that govern the relationships between signature(s) and countersignature(s) and that dictate when they should be accepted as valid, are out of the scope of the present document as they will likely be established specifically for each scenario.

## G.2.2.8 Checking `DataObjectFormat`

The verifier should check that the `ObjectReference` element actually references one `ds:Reference` element from the signature.

In addition, should this property refer to a `ds:Reference` that in turn refers to a `ds:Object`, the verifier should check that attributes `MimeType` and `Encoding` are equal in both elements.

Additional rules governing the acceptance of the XAdES signature as valid or not in the view of the contents of this property are out of the scope of the present document.

### G.2.2.9 Checking `CommitmentTypeIndication`

The verifier should check that all the `ObjectReference` elements actually reference `ds:Reference` elements from the signature.

Additional rules governing the acceptance of the XAdES signature as valid or not in the view of the contents of this property are out of the scope of the present document.

### G.2.2.10 Checking `SignatureProductionPlace`

Specific rules governing the acceptance of the XAdES signature as valid or not in the view of the contents of this property are out of the scope of the present document.

### G.2.2.11 Checking `SignerRole`

Should this property contain claimed roles, the specific rules governing the acceptance of the XAdES signature as valid or not in the view of the contents of this property are out of the scope of the present document.

If this property contains some certified role, the verifier should verify the validity of the attribute certificates present. Additional rules the governing the acceptance of the XAdES signature as valid or not in the view of the contents of this property, are out of the scope of the present document.

### G.2.2.12 Checking `CompleteCertificateRefs` and `AttributeCertificateRefs`

If `CompleteCertificateRefs` is present the verifier should:

1) Gain access to all the CA certificates that are part of the certification path, according to what has been stated in clause G.2.2.2.

2) Check that for each certificate in the aforementioned set, the property contains its corresponding reference. For doing this the values of the `IssuerSerial`, `ds:DigestMethod` and `ds:DigestValue` should be checked as indicated in clause G.2.2.5 steps 1 and 3.

3) Check that there are no references to certificates out of those that are part of the certification path.

Rules for `AttributeCertificateRefs` are similar but instead CA certificates in the certification path of the signature, the checks will be on the set containing all the Attribute Authorities certificates that are used for validating the attribute certificate present in the signature.

### G.2.2.13 Checking `CompleteRevocationRefs` and `AttributeRevocationRefs`

Checking `CompleteRevocationRefs` requires that the verifier gains access to information of the status of all the certificates used for the verification of the signature. Clause G.2.2.3 discusses different means that the verifier may use for doing this. If this `CompleteRevocationRefs` is present the verifier should perform the following steps:

1) If `RevocationValues` is present, the verifier should check that they actually provide adequate revocation information for all the certificates required for verifying the electronic signature. If so, the verifier should check the references in `CompleteRevocationRefs` against the values in `RevocationValues` proceeding as indicated in step 3.

2) If `RevocationValues` is not present, the verifier should get the revocation information data referenced in the property and check if they actually provide adequate revocation information for all the certificates required for verifying the electronic signature. If so, the verifier should check the references against this data proceeding as indicated in step 3.

3) If the verifier has retrieved CRLs, check that each CRL correctly matches its reference within RevocationRefs. For doing this, for each CRL:

- If there is no CRLRefs element, the verifier should treat this signature as invalid. If there is a non-empty list, take the first CRLRef element in the list and:

  a) Check that the string format of the issuer"s DN of the CRL generated as stated in XMLDSIG, is the same as the value present in the Issuer element.

  b) Check that the time indicated by the thisUpdate field in the CRL is the same as the time indicated by the IssueTime element.

  c) If the CRL contains the cRLNumber extension, check that its value is the same as the value indicated by the Number element.

  d) If the aforementioned checks are successful, compute the digest of the CRL according to the algorithm indicated in the ds:DigestMethod element, base64 encode the result and check if this is the same as the contents of the ds:DigestValue element.

- If any of these checks fails, repeat the process for the next CRLRef elements until finding one satisfying them or finishing the list. If none of the references matches the CRL, the verifier should treat the signature as invalid.

1) If the verifier has retrieved OCSP responses, check that each OCSP response correctly matches its reference within RevocationRefs. For doing this, for each OCSP response:

- If there is no OCSPRefs element, the verifier should treat this signature as invalid. If there is a list and is not empty, take the first OCSPRef element in the list and:

  a) Check that the content of ResponderID element matches the content of the responderID field within the OCSP response. If the content of this field is the byName choice, check if its string format is the same. If the content of this field is the byKey choice, the ResponderID should contain the base64 encoded key digest. The verifier should check if this value matches the byKey choice.

  b) Check that the time indicated by the thisUpdate field in the OCSP response is the same as the time indicated by the ProducedAt element.

  c) If the aforementioned checks are successful, compute the digest of the OCSP response according to the algorithm indicated in the ds:DigestMethod element, base64 encode the result and check if this is the same as the contents of the ds:DigestValue element.

- If any of the checks fails, repeat the process for the next OCSPRef elements until finding one satisfying them or finishing the list. If none of the references matches the OCSP response, the verifier should treat the signature as invalid.

1) Check that there are no CRLRef elements referencing other CRLs than those that have been retrieved in steps 1 or 2.

2) Check that there are no OCSPRef elements referencing other OCSP responses than those that have been retrieved in steps 1 or 2.

Rules for AttributeRevocationRefs are similar but instead of revocation data of certificates used for verifying the signature, the checks will be on the set of all the revocation data that is used for validating the attribute certificate present in the signature.

## G.2.2.14 Checking CertificateValues and AttrAuthoritiesValues

Clause G.2.2.2 mandates that if CertificateValues is present, the verifier should check that the certificates present there and within ds:KeyInfo, actually are all the required certificates for verifying the signature. It also states that if AttrAuthoritiesValues is present, the verifier should check that the certificates present there, and within the CertificateValues, actually are all the required certificates for validating the attribute certificate present in the signature.

In addition, clauses G.2.2.5 and G.2.2.12 have stated rules for the jointly usage of this property with SigningCertificate and CompleteCertificateRefs and the AttributeCertificateRefs.

Additional rules governing the acceptance of the XAdES signature as valid or not in the view of the contents of this property are out of the scope of the present document.

### G.2.2.15 Checking `RevocationValues` and `AttributeRevocationValues`

Clause G.2.2.3 mandates that if `RevocationValues` is present, the verifier should check that the revocation information data present there actually provide adequate revocation information for all the certificates required for verifying the signature. In addition, if `AttributeRevocationValues` is present, the verifier should check that the revocation data present there actually provide adequate revocation information for all the certificates required for validating the attribute certificate present in the XAdES signature.

Clause G.2.2.13 has stated rules for the join usage of this property with `CompleteRevocationRefs` and `AttributeRevocationRefs`.

In addition the verifier should check that the status of the certificates reported in these data is VALID.

Additional rules governing the acceptance of the XAdES signature as valid or not in the view of the contents of this property are out of the scope of the present document.

### G.2.2.16 Checking time-stamp tokens

The present clause presents the rules that should govern the verification of the time-stamp tokens encapsulated within the containers defined by the present specification.

As stated in the normative part, the present document allows using two mechanisms for identifying what data objects are being time-stamped by the time-stamp token enclosed in a container property, namely implicit and explicit. The normative part details which properties and under which circumstances to use the explicit mechanism.

Certain containers are specified to use only one mechanism; others are specified to use one or the other depending on the circumstances, and that is why this clause is divided in two clauses, one for each group of containers.

Time-stamp tokens may contain accuracy information, ie, an indication of the time deviation around the time included in the token. Rules dictating how to use this information when taking the decision whether a certain time is previous or ulterior to the time within the time-stamp token, are a matter of policy and in consequence, out of the scope of the present specification. This applies to any mention done hereinafter to a comparison of time values where a time within a time-stamp token is involved.

#### G.2.2.16.1 Containers using one identification mechanism

##### G.2.2.16.1.1 Checking `AllDataObjectsTimeStamp`

The time-stamp token contained within this property does not cover any unsigned property and the regular elements within the signature that are mandated to be time-stamped are easily determined by inspecting the `ds:SignedInfo` contents. That is why this container will exclusively use the implicit mechanism.

The verifier should perform the following steps:

1) Verify the signature present within the time-stamp token. Rules for acceptance of the validity of the signature within the time-stamp, involving trust decisions, are out of the scope of the present document.

2) Take, the first `ds:Reference` element within `ds:SignedInfo` if and only if the `Type` attribute doesn"t have the value "http://uri.etsi.org/01903#SignedProperties".

3) Process it according to the reference processing model of XMLDSIG.

4) If the result is a node-set, canonicalize it using the algorithm indicated in `CanonicalizationMethod` element of the property, if present. If not, the standard canonicalization method as specified by XMLDSIG must be used.

5) Concatenate the resulting bytes in an octet stream.

6) Repeat steps 2 to 4 for all the subsequent `ds:Reference` elements (in their order of appearance) within `ds:SignedInfo` if and only if `Type` attribute has not the value "http://uri.etsi.org/01903#SignedProperties".

7) For each time-stamp token encapsulated by the property, compute the digest of the resulting octet stream using the algorithm indicated in the time-stamp token and check if it is the same as the digest present there.

8) Check for coherence in the value of the times indicated in the time-stamp tokens. All the time instants must be previous to the time when the verification is being made, to the time indicated within the `SigningTime` if present, and to the times indicated within the time-stamp tokens enclosed within all the rest of time-stamp container properties except `IndividualDataObjectsTimeStamp`.

#### G.2.2.16.1.2 Checking IndividualDataObjectsTimeStamp

As before, the time-stamp token contained within this property does not cover any unsigned property, but now there is need for explicit information of what of the signed data-objects are actually time-stamped. The consequences is that it will exclusively use the explicit mechanism.

The verifier should perform the following steps:

1) Verify the signature present within the time-stamp token. Rules for acceptance of the validity of the signature within the time-stamp, involving trust decisions, are out of the scope of the present document.

2) Take the first `Include` element.

3) Check the coherence of the value of the not-fragment part of the URI within its `URI` attribute according to the rules stated in clause 7.1.4.3.1.

4) De-reference the URI according to the rules stated in clause 7.1.4.3.1.

5) Check that the retrieved element is actually a `ds:Reference` element of the `ds:SignedInfo` of the qualified signature and that its `Type` attribute (if present) does not have the value "http://uri.etsi.org/01903#SignedProperties".

6) Process it according to the reference processing model of XMLDSIG.

7) If the result is a node-set, canonicalize it using the algorithm indicated in `CanonicalizationMethod` element of the property, if present. Otherwise use the standard canonicalization method as specified by XMLDSIG.

8) Concatenate the resulting bytes in an octet stream.

9) Repeat steps 2 to 4 for all the subsequent `Include` elements (in their order of appearance) within the time-stamp token container.

10) For each time-stamp token encapsulated by the property, compute the digest of the resulting byte stream using the algorithm indicated in the time-stamp token and check if it is the same as the digest present there.

11) Check for coherence in the value of the times indicated in the time-stamp tokens. All the time instants must be previous to the time when the verification is being made, and to the times indicated within the time-stamp tokens enclosed within all the rest of time-stamp container properties except `AllDataObjectsTimeStamp`.

#### G.2.2.16.1.3 Checking `SignatureTimeStamp`

As stated by clause 7.3, this container envelopes a time-stamp token on the `ds:SignatureValue` element and exclusively uses the implicit mechanism.

The verifier should perform the following steps:

1) Verify the signature present within the time-stamp token. Rules for acceptance of the validity of the signature within the time-stamp, involving trust decisions, are out of the scope of the present document.

2) Take the `ds:SignatureValue` element.

3) Canonicalize it using the algorithm indicated in `CanonicalizationMethod` element of the property, if present. Otherwise use the standard canonicalization method as specified by XMLDSIG.

4) For each time-stamp token encapsulated by the property, compute the digest of the resulting byte stream using the algorithm indicated in the time-stamp token and check if it is the same as the digest present there.

5) Check for coherence in the values of the times indicated in the time-stamp tokens. They must be posterior to the one indicated in the `SigningTime` property, and to the times indicated in the time-stamp tokens contained within `AllDataObjectsTimeStamp` or `IndividualDataObjectsTimeStamp`, if present. Finally they must be previous to the times indicated in the time-stamp tokens enclosed by any `RefsOnlyTimeStamp`, `SigAndRefsTimeStamp` or/and `ArchiveTimeStamp` present elements.

## G.2.2.16.2 Containers using two/both identification mechanism

The rest of containers, namely `SigAndRefsTimeStamp`, `RefsOnlyTimeStamp` and `ArchiveTimeStamp` may use both mechanisms.

First of all, a set of common rules for all these containers will be given, and afterwards, there will be specific clauses that will particularize these rules for each property.

### G.2.2.16.2.1 Common rules

When a time-stamp container incorporates `Include` elements (explicit identification mechanism), the verifier should perform the following steps:

1) Verify the signature present within the time-stamp token. Rules for acceptance of the validity of the signature within the time-stamp, involving trust decisions, are out of the scope of the present document.

2) Check that all the signed properties and not property elements of the signature that are mandated to be time-stamped by the normative clause defining the time-stamp token container are present.

3) Take each one of the signed properties and not property elements in the signature that the normative part dictates that must be time-stamped, in the order specified in the normative clause defining the time-stamp token container type. Canonicalize them and concatenate the resulting bytes in one octet stream. If the `CanonicalizationMethod` element of the property is present, use it for canonicalizing. Otherwise, use the standard canonicalization method as specified by XMLDSIG.

4) For each `Include` element:

    a) Check the coherence of the value of the not-fragment part of the URI within its `URI` attribute according to the rules stated in clause 7.1.4.3.1.

    b) De-reference the URI according to the rules stated in clause 7.1.4.3.1.

    c) Check that the retrieved element must actually be covered by the time-stamp token according to the specification.

    d) Proceed to process the data retrieved as specified in clause 7.1.4.3.1 and to concatenate the result to the octet stream mentioned in step 2.

5) Check that all the Include elements actually refer to all the existing unsigned properties in XAdES signature that must be covered by the time-stamp token.

6) For each time-stamp token encapsulated by the property, compute the digest of the resulting byte stream using the algorithm indicated in the time-stamp token and check if it is the same as the digest present there.

7) Check for coherence in the values of the times indicated in the time-stamp tokens. This implies that the indicated time instants must be previous to the time when the verification is being made, and also certain specific rules for each time-stamp token container type.

When a time-stamp container does not incorporates Include elements (implicit identification mechanism), the verifier should perform the following steps:

1) 1) Verify the signature present within the time-stamp token. Rules for acceptance of the validity of the signature within the time-stamp, involving trust decisions, are out of the scope of the present document.

2) 2) Check that all the signed properties and regular elements of the signature that are mandated to be time-stamped by the normative clause defining the time-stamp token container, are present.

3) Take each one of the signed properties and not property elements in the signature that the normative part dictates that must be time-stamped, in the order specified in the normative clause defining the time-stamp token container type. Canonicalize them and concatenate the resulting bytes in one octet stream. If the `CanonicalizationMethod` element of the property is present, use it for canonicalizing. Otherwise, use the standard canonicalization method as specified by XMLDSIG.

4) Check that all the unsigned properties that are mandated to be time-stamped by the normative clause defining the time-stamp token container, are present in the XAdES signature.

5) Take each of the unsigned properties that appear BEFORE the time-stamp token container within the `UnsigedSignatureProperties` element if and only if that property is mandated to be time-stamped by the normative clause defining such time-stamp token container. Canonicalize them and concatenate these results to the aforementioned octet stream.

6) For each time-stamp token encapsulated by the property, compute the digest of the resulting byte stream using the algorithm indicated in the time-stamp token and check if it is the same as the digest present there.

7) Check for coherence in the values of the times indicated in the time-stamp tokens. This implies that the indicated time instants must be previous to the time when the verification is being made, and also certain specific rules for each time-stamp token container type.

#### G.2.2.16.2.2 Checking `RefsOnlyTimeStamp`

The following particular rules apply for this property no matter whether implicit or explicit mechanism is used:

1) Steps 2 and 3 of both cases do not apply, as the enclosed time-stamp tokens just cover unsigned properties of the signature.

2) In step 7 of both cases, the verifier should check that all the times indicated by the time-stamp tokens in the property are posterior to the one indicated in the `SigningTime` property, and to the times indicated in the time-stamp tokens contained within `AllDataObjectsTimeStamp`, `IndividualDataObjectsTimeStamp` or `SignatureTimeStamp`, if present. They must also be previous to the times indicated in the time-stamp tokens enclosed by any `ArchiveTimeStamp` present elements.

The following particular rules apply when explicit mechanism is used:

1) In step 4.c the verifier should check that each retrieved element is one of the following properties in the signature: `CompleteCertificateRefs`, `CompleteRevocationRefs`, `AttributeCertificateRefs` and `AttributeRevocationRefs`.

2) In step 5, the verifier should check that all the following properties: `CompleteCertificateRefs`, `CompleteRevocationRefs` and, if present, `AttributeCertificateRefs` and `AttributeRevocationRefs`, have actually been retrieved.

The following particular rules apply when implicit mechanism is used:

1) In step 4 the verifier should check that the `CompleteCertificateRefs` and `CompleteRevocationRefs` properties are present in the signature. She should also check that they and, if present, `AttributeCertificateRefs` and `AttributeRevocationRefs,` appear before `RefsOnlyTimeStamp`.

2) In step 5, the verifier should take `CompleteCertificateRefs` and `CompleteRevocationRefs` and if present, `AttributeCertificateRefs` and `AttributeRevocationRefs` in their order of appearance.

#### G.2.2.16.2.3 Checking `SigAndRefsTimeStamp`

The following particular rules apply for this property no matter whether implicit or explicit mechanism is used:

1) In step 3, the verifier should take `ds:SignatureValue` and process it as indicated in the aforementioned step.

2) In step 7 of both cases, the verifier should check that all the times indicated by the time-stamp tokens in the property are posterior to the one indicated in the `SigningTime` property, and to the times indicated in the time-stamp tokens contained within `AllDataObjectsTimeStamp`, `IndividualDataObjectsTimeStamp` or `SignatureTimeStamp`, if present. They must also be previous to the times indicated in the time-stamp tokens enclosed by any `ArchiveTimeStamp` present elements.

The following particular rules apply when explicit mechanism is used:

1) In step 4.c the verifier should check that each retrieved element is one of the following elements in the signature: `ds:SignatureValue`, `CompleteCertificateRefs`, `SignatureTimeStamp`, `CompleteRevocationRefs`, `AttributeCertificateRefs` and `AttributeRevocationRefs`.

2) In step 5, the verifier should check that all the following elements: `ds:SignatureValue`, `CompleteCertificateRefs`, `CompleteRevocationRefs` and, if present, `SignatureTimeStamp`, `AttributeCertificateRefs` and `AttributeRevocationRefs`, have actually been retrieved.

The following particular rules apply when implicit mechanism is used:

1) In step 4 the verifier should check that the `CompleteCertificateRefs` and `CompleteRevocationRefs` properties are present in the signature. She should also check that they and, if present, `SignatureTimeStamp`, `AttributeCertificateRefs` and `AttributeRevocationRefs`, appear before `SigAndRefsTimeStamp`.

2) In step 5, the verifier should take the aforementioned properties in their order of appearance within the signature.

## G.2.2.16.2.4 Checking `ArchiveTimeStamp`

The following particular rules apply for this property no matter whether implicit or explicit mechanism is used.

1) Substitute step 3 by the process indicated here. The verifier should take the following elements in the order they are listed: all the `ds:SignedReference` in the order they appear within `ds:SignedInfo`, `ds:SignedInfo`, `ds:SignatureValue`, and `ds:KeyInfo` if present. Afterwards, the verifier should process each `ds:Reference` according to the reference processing model of XMLDSIG, canonicalize the result and concatenate the resulting bytes in the final octet stream. After that the verifier should canonicalize the rest of the elements and concatenate the resulting bytes to the final octet stream.

2) In step 7, the verifier should check that all the times indicated by the time-stamp tokens in the property are posterior to the one indicated in the `SigningTime` property, and to the times indicated in the time-stamp tokens contained within `AllDataObjectsTimeStamp, IndividualDataObjectsTimeStamp`, `SignatureTimeStamp` if present, and `RefsOnlyTimeStamp` or `SigAndRefsTimeStamp`, if present They must also be previous to the times indicated in the time-stamp tokens enclosed by any `ArchiveTimeStamp` that appear before the one that is being verified.

The following particular rules apply when explicit mechanism is used:

1) In step 4.c the verifier should check that the retrieved element is one of the following signature elements: `SignatureTimeStamp`, `CounterSignature`, `CompleteCertificateRefs`, `CompleteRevocationRefs`, `AttributeCertificateRefs`, `AttributeRevocationRefs`, `CertificateValues`, `RevocationValues`, `SigAndRefsTimeStamp`, `RefsOnlyTimeStamp`, `ArchiveTimeStamp` or any of the `ds:Object` present elements not referenced in some `ds:Reference`.

2) In step 5, the verifier should check that the retrieved elements are: all the existing `SignatureTimeStamp`, all the present `CounterSignature`, the `CompleteCertificateRefs`, and the `CompleteRevocationRefs` if present, the `AttributeCertificateRefs` and `AttributeRevocationRefs` if present, `CertificateValues` and `RevocationValues`, any existing `SigAndRefsTimeStamp` or `RefsOnlyTimeStamp`, all the `ds:Object` present elements not referenced in some `ds:Reference`, and certain `ArchiveTimeStamp`.

The following particular rules apply when implicit mechanism is used.

1) In step 4 the verifier should check that the `CertificateValues` and `RevocationValues` properties are present in the signature. If one of them is not present, the verifier should take it as a failure in the verification process.

2) Step 5 is performed as indicated. The verifier will take, among the unsigned properties that appear before the property that is being verified, those that appear in the following list, and in their order of appearance: `SignatureTimeStamp`, `CounterSignature`, `CompleteCertificateRefs`, `CompleteRevocationRefs`, `AttributeCertificateRefs`, `AttributeRevocationRefs`, `CertificateValues`, `RevocationValues`, `SigAndRefsTimeStamp`, `RefsOnlyTimeStamp`, `ArchiveTimeStamp` and all the `ds:Object` elements that are not covered by some `ds:Reference`.

# Annex H (informative): Bibliography

- Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures.
- IETF RFC 2634: "Enhanced Security Services for S/MIME".
- IETF RFC 2396: "Uniform Resource Identifiers (URI): Generic Syntax".
- IETF RFC 3061: "A URN Namespace of Object Identifiers".
- W3C 12-2000 (W3C Candidate Recommendation, December 2000): "The Platform for Privacy Preferences 1.0 (P3P1.0) Specification".

# History

| Document history | | |
|---|---|---|
| V1.1.1 | February 2002 | Publication |
| V1.2.1 | March 2004 | Publication (Withdrawn) |
| V1.2.2 | April 2004 | Publication |
| V1.3.2 | March 2006 | Publication |
| | | |