Draft EN 319 122-1 V0.0.3 (2013-11)

Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES); Part 1: Core Specification

Reference

DEN/ESI-0019122-1

Keywords

ASN.1, electronic signature, profile, 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 draft European Standard (EN) has been produced by ETSI Technical Committee Electronic Signatures and Infrastructures (ESI) and is now submitted for the combined Public Enquiry and Vote phase of the ETSI standards EN Approval Procedure.

The present document is part 1 of a multi-part deliverable covering the CMS Advanced Electronic Signatures (CAdES), as identified below:

Part 1: Core Specification

Part 2: CAdES Baseline Profile

The present document was previously published as TS 101 733 [i.1].

Proposed national transposition dates	
Date of latest announcement of this EN (doa):	3 months after ETSI publication
Date of latest publication of new National Standard or endorsement of this EN (dop/e):	6 months after doa
Date of withdrawal of any conflicting National Standard (dow):	6 months after doa

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 [i.16] 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 present document:

- presents attributes defined in defines which are specified in RFC 5652[7], RFC 2634 [2], and the present document that can be use to create an advanced electronic signature based on CMS.
- specifies formats for CMS advanced electronic signatures that, by using the aforementioned attributes, remain valid over long periods and incorporate additional useful information in common use cases.
- defines a set of conformance requirements to claim endorsement to the present document.

The present document specifies two main types of attributes: signed attributes and unsigned attributes. The first ones are attributes that are also covered by the signature produced by the signer and stored in the `SignerInfo` element, which implies that the signer has these attributes before creating the signature. The unsigned attributes are 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 `SignerInfo` element (the one computed by the signer); however they can be actually covered by subsequent times

EDITOR NOTE: a number of editor notes like this one appears throughout the document. These notes intend to attract readers' attention and/or kindly request their feedback on certain specific issues.

EDITOR NOTE: the annex for the hash computation for the time-stamps was removed, to avoid any inconsistencies now and in the future between the normative text and the informative annex. The normative clauses on the attributes containing time-stamps should be clear enough to understand. We kindly ask the stakeholder to signal us if some descriptions in the corresponding clauses are not clear enough.

1 Scope

The present document describes formats for advanced electronic signatures using ASN.1 (Abstract Syntax Notation 1) 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. These formats are based on CMS (Cryptographic Message Syntax) defined in RFC 5652 [7]. These electronic signatures are thus called CAdES, for "CMS Advanced Electronic Signatures".

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 trust 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:

- shows a taxonomy of the qualifying information (attributes) whose presence in an electronic signature allows it to remain valid over long periods, to satisfy common use cases requirements, and to be compliant with the European Directive;
- defines a number of electronic signature formats and the corresponding conformances levels, including electronic signatures that can remain valid over long periods. This includes evidence as to its validity even if the signer or verifying party later attempts to deny (repudiates) the validity of the electronic signature;

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

Clause 5 lists some types and data structures that are used in the attributes defined in clause 6.

Clause 6 defines the qualifying attributes except those attributes that contain references to validation data.

Clause 7 defines different conformance levels that may be claimed against the present document.

Normative Annex A defines qualifying attributes that encapsulate references to validation data, attributes that encapsulate time-stamp tokens for these references, and attributes that have been obsoleted by new attributes (for preserving management of legacy electronic signatures).

Normative Annex B defines CAdES forms that incorporate attributes encapsulating references to validation data and attributes encapsulating time-stamp tokens for these references.

Normative Annex C specifies conformance levels for CAdES signatures that incorporate attributes encapsulating references to validation data and attributes encapsulating time-stamp tokens for these references.

Informative Annex D provides rationale for the attributes and mechanisms specified by this document.

Normative Annex E provides a summary of all the ASN.1 syntax definitions for new syntax defined in the present document.

Informative Annex F shows an example structured content and MIME.

Informative Annex G shows changes from previous versions.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

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

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

[1] IETF RFC 2045 (1996): "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies".

[2] IETF RFC 2634 (1999): "Enhanced Security Services for S/MIME".

[3] IETF RFC 3161 (2001): "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)".

[4] IETF RFC 4998 (2007): "Evidence Record Syntax (ERS)".

[5] IETF RFC 5035 (2007): "Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility".

[6] IETF RFC 5280 (2008):"Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile".

NOTE: Obsoletes RFC 3280

[7] IETF RFC 5652 (2009): "Cryptographic Message Syntax (CMS)"

NOTE: Obsoletes RFC 3852

[8] IETF RFC 5755 (2010): "An Internet Attribute Certificate Profile for Authorization"

NOTE: Obsoletes RFC 3281

[9] IETF RFC 6960 (2013):" X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP".

NOTE: Obsoletes RFC 2660

[10] Recommendation ITU-T X.208 (1988): "Specification of Abstract Syntax Notation One (ASN.1)".

NOTE: Recommendation ITU-T X.208 has been withdrawn on 30 October 2002 as it has been superseded by Recommendations ITU-T X.680-683. All known defects in X.208 have been corrected in Recommendations ITU-T X.680-683 (1993) further revised in 1997 and 2002. However, the reference is kept in the current to ensure compatibility with RFC 5652 [7].

[11] Recommendation ITU-T X.680 (2008): "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

[12] Recommendation ITU-T X.500: "Information technology - Open Systems Interconnection - The Directory: Overview of concepts, models and services".

[13] Recommendation ITU-T X.501 (2008)/ISO/IEC 9594-1 (2008): "Information technology - Open Systems Interconnection - The Directory: Models".

[14] Recommendation ITU-T X.509 (2008)/ISO/IEC 9594-8 (2008): "Information technology - Open Systems Interconnection - The Directory: Public-key and Attribute Certificate frameworks".

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

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

[i.2] ETSI TS 101 861: "Electronic Signatures and Infrastructures (ESI); Time stamping profile".

[i.3] ETSI TR 102 272: "Electronic Signatures and Infrastructures (ESI); ASN.1 format for signature policies".

[i.4] ETSI TR 119 001: "Electronic Signatures and Infrastructures (ESI); Definitions and abbreviations".

[i.5] ETSI TS 319 102: "Electronic Signatures and Infrastructures (ESI); Procedures for Signature Creation and Validation".

[i.6] ETSI EN 319 122-2: "Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES); Part 2: Baseline Profile"

[i.7] ETSI EN 319 132-1 "Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES); Part 1: XML Advanced Electronic Signatures (XAdES)"

[i.8] ETSI EN 319 172-1: "Electronic Signatures and Infrastructures (ESI); Signature Policies; Part 1: Concept and common requirements".

[i.9] ETSI EN 319 172-2: "Electronic Signatures and Infrastructures (ESI); Signature Policies; XML format for signature policies".

[i.10] ETSI EN 319 172-3: "Electronic Signatures and Infrastructures (ESI); Signature Policies; ASN.1 format for Signature Policies".

[i.11] ETSI EN 319 411-2: "Electronic Signatures and Infrastructures (ESI); Policy and security requirements for Trust Service Providers issuing certificates; Part 2: Policy requirements for certification authorities issuing qualified certificates".

[i.12] ETSI EN 319 411-3: "Electronic Signatures and Infrastructures (ESI); Policy and security requirements for Trust Service Providers issuing certificates; Part 3: Policy requirements for certification authorities issuing public key certificates".

[i.13] ETSI EN 319 411-4: "Electronic Signatures and Infrastructures (ESI); Policy and security requirements for Trust Service Providers issuing certificates; Part 4: Policy requirements for certification authorities issuing attribute certificates".

[i.14] ETSI EN 319 412-2: "Electronic Signatures and Infrastructures (ESI); Profiles for Trust Service Providers issuing certificates; Part 2: Certificate Profile for certificates issued to natural persons".

[i.15] ETSI EN 319 412-3: "Electronic Signatures and Infrastructures (ESI); Profiles for Trust Service Providers issuing certificates; Part 3: Certificate profile for certificates issued to legal persons".

[i.16] Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures.

[i.17] IETF RFC 2479 (1998): "Independent Data Unit Protection Generic Security Service Application Program Interface (IDUP-GSS-API)".

[i.18] IETF RFC 2743 (2000): "Generic Security Service Application Program Interface Version 2, Update 1".

[i.19] IETF RFC 3125 (2000): "Electronic Signature Policies".

[i.20] IETF RFC 3494 (2003): "Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2".

[i.21] IETF RFC 3851: "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification".

[i.22] IETF RFC 4523 (2006): "Lightweight Directory Access Protocol (LDAP). Schema Definitions for X.509 Certificates".

[i.23] IETF RFC 4210 (2005): "Internet X.509 Public Key Infrastructure Certificate Management Protocols".

[i.24] ISO 7498-2 (1989): "Information processing systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture".

[i.25] ISO/IEC 10181-5 (September 1996): "Information technology - Open Systems Interconnection - Security frameworks for open systems: Confidentiality framework".

[i.26] ISO/IEC 13888-1 (2004): "IT security techniques - Non-repudiation - Part 1: General".

[i.27] CWA 14171:2004: "General guidelines for electronic signature verification".

[i.28] “Security Assertion Markup Language (SAML) v1.1” (2003)

[i.29] “Security Assertion Markup Language (SAML) v2.0” (2005)

NOTE: The documents [i.4], [i.5], [i.6], [i.7], [i.8], [i.9], [i.10], [i.11], [i.12], [i.13], [i.14], [i.15] are published in the context of the work in Mandate M460. They might not yet be published.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in TR 119 001 [i.4] and the following apply:

arbitrator: Entity that arbitrates a dispute between a signer and verifier when there is a disagreement on the validity of an electronic signature

Attribute Authority (AA): authority that assigns privileges by issuing attribute certificates

Attribute Certificate Revocation List (ACRL): revocation list containing a list of references to attribute certificates that are no longer considered valid by the issuing authority

authority certificate: certificate issued to an authority (e.g. either to a certification authority or an attribute authority)

Certification Authority (CA): authority trusted by one or more users to create and assign public key certificates; optionally, the certification authority may create the users' keys

NOTE: See Recommendation ITU-T X.509 [14].

Certificate Revocation List (CRL): signed list indicating a set of public key certificates that are no longer considered valid by the certificate issuer

delta CRL: lists those certificates, within its scope, whose revocation status has changed since the issuance of a referenced complete CRL, see RFC 5280 [6].

digital signature: data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery, e.g. by the recipient

NOTE: See ISO 7498-2 [i.24].

electronic signature: data in electronic form that is attached to or logically associated with other electronic data and that serves as a method of authentication

NOTE: See Directive 1999/93/EC [i.16] of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures.

Explicit Policy-based Electronic Signature (EPES): electronic signature where the signature policy to be used to validate it is explicitly specified

extended electronic signatures: electronic signatures enhanced by incorporation of validation data such as time-stamp tokens and certificate revocation data, to address commonly recognized threats

grace period: time period that permits the certificate revocation information to propagate through the revocation process to relying parties

Public Key Certificate (PKC): public key of a user, together with some other information, rendered unforgeable by digital signature with the private key of the certification authority which issued it

NOTE: See Recommendation ITU-T X.509 [14].

relying party: recipient of a certificate who acts in reliance on that certificate and/or digital signatures verified using that certificate

signature policy: set of rules for the creation and validation of one (or more interrelated) electronic signature(s) that defines the technical and procedural requirements for creation, validation and (long term) management of this (those) electronic signature(s), in order to meet a particular business need, and under which the signature(s) can be determined to be valid.

signer: entity that creates an electronic signature

time-mark: information in an audit trail from a Trust Service Provider that binds a representation of a datum to a particular time, thus establishing evidence that the datum existed before that time

time-marking authority: trusted third party that creates records in an audit trail in order to indicate that a datum existed before a particular point in time

time-stamp token: data object that binds a representation of a datum to a particular time, thus establishing evidence that the datum existed before that time

Time-Stamping Authority (TSA): trusted third party that creates time-stamp tokens in order to indicate that a datum existed at a particular point in time

Time-Stamping Unit (TSU): set of hardware and software that is managed as a unit and has a single time-stamp token signing key active at a time

Trust Service Provider (TSP): entity that helps to build trust relationships by making available or providing some information upon request

validation data: additional data, collected by the signer and/or a verifier, needed to validate the electronic signature

NOTE: It may include: certificates, revocation status information (such as CRLs or OCSP-Responses), time-stamps or time-marks.

verifier: entity that wants to validate or verify an electronic signature

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in TR 119 001 [i.4] and the following apply:

AA	Attribute Authority
ACRL	Attribute Certificate Revocation List
API	Application Program Interface
ASCII	American Standard Code for Information Interchange
ATSv2	archive-time-stamp attribute

NOTE: As defined in clause B.4.

ATSv3	archive-time-stamp-v3 attribute

NOTE: As defined in clause 6.5.2.

BER	Basic Encoding Rules
BES	Basic Electronic Signature
CA	Certification Authority
CAdES	CMS Advanced Electronic Signature
CAdES-A	CAdES with Archive validation data
CAdES-BES	CAdES Basic Electronic Signature
CAdES-C	CAdES with Complete validation data
CAdES-EPES	CAdES Explicit Policy Electronic Signature
CAdES-LT	CAdES Long Term
CAdES-T	CAdES with Time
CAdES-X Long	CAdES with Extended Long validation data
CAdES-X	CAdES with eXtended validation data
CAdES-XL	CAdES –X Long
CMS	Cryptographic Message Syntax
CRL	Certificate Revocation List
DER	Distinguished Encoding Rules (for ASN.1)
EDIFACT	Electronic Data Interchange For Administration, Commerce and Transport
EPES	Explicit Policy-based Electronic Signature
ES	Electronic Signature
ESS	Enhanced Security Services (enhances CMS)
GSM	Global System for Mobile Communications
LT	Long-Term
MIME	Multipurpose Internet Mail Extensions
OCSP	Online Certificate Status Provider
OID	Object IDentifier

PKC	Public Key Certificate
SHA-1	Secure Hash Algorithm 1
SIM	Subscriber Identity Module
TSA	Time-Stamping Authority
TSP	Trust Service Provider
TSU	Time-Stamping Unit
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XAdES	XML Advanced Electronic Signatures
XML	Extensible Markup Language

4 Overview

The present document defines a number of Electronic Signature (ES) formats that build on CMS (RFC 5652 [7]) by adding signed and unsigned attributes. These attributes are described in the present document and are either defined in CMS (RFC 5652 [7]), ESS (RFC 2634 [2] and RFC 5035 [5]) or the present document. Below follows a short overview of the used attributes:

- General CMS signature attributes. These attributes are needed to create the basic CMS electronic signature.
    - `content-type`. It is defined in RFC 5652 [7] and specifies the type of the `EncapsulatedContentInfo` value being signed. Details are provided in clause 6.1.1.
    - `message-digest`. It is defined in RFC 5652 [7] and contains the message digest of the `eContent` within the `encapContentInfo` being signed. Details are provided in clause 6.1.2
- Signing certificate reference attribute. A signing certificate reference attribute provides an unambiguous way of identifying the certificate used for the signature and is necessary to create a CMS Advanced Electronic Signature. See clause 6.2.2 for details of these attributes.
    - `ESS signing-certificate`, as defined in RFC 2634: "Enhanced Security Services" [2] (ESS hereafter), only allows for the use of SHA-1 as digest algorithm.
    - `ESS signing-certificate-v2`, as defined in RFC 5035: "ESS Update: Adding CertID Algorithm Agility" [5], allows for the use of any digest algorithm.
- `signature-policy-identifier`. This attribute provides an unambiguous way of 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 validation process. The 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 the signer implied a different signature policy. Details on this property can be found in clause 6.2.9.
- Validation data attributes. Validation data (certificates chains, CRLs, OCSP responses, etc.) can be stored directly in within the root `SignedData.certificates`, or `SignedData.crls`, as recommended for CAdES-A with ATSv3. The following attributes provide alternative ways to store the validation data:
    - `certificate-values`: this attribute is defined by the present document. It encapsulates certificates used in the validation of the electronic signature. See clause A.1.1.2 for details.
    - `revocation-values`: this attribute is defined by the present document. It contains CRLs and/OCSP responses required for the validation of the signature, see clause A.1.2.2 for details

  And references to validation data:

    - `complete-certificate-references`: this attribute is defined by the present document. It contains references to certificates used in the validation of the electronic signature. See clause A.1.1.1 for further details.
    - `complete-revocation-references`: this attribute is defined by the present document. It contains references to CRLs and/or OCSP responses used for verifying the signature. See clause A.1.2.1 for details.
    - `attribute-certificate-references`: this attribute is defined by the present document. It contains references to certificates of the AA or any other CA used in the validation of an attribute certificate or a signed assertion, see clause A.1.3 for details.
    - `attribute-revocation-references`: this attribute is defined by the present document. It contains references to ACRLs, CRLs or OCSP responses that have been used in the validation of a attribute certificate or a signed assertion, see clause A.1.4 for details.

- `content-time-stamp`: this attribute is defined by the present document. It allows incorporating within the signed information a time-stamp token of the data to be signed. It provides proof of the existence of the data before the signature was created. Clause 6.2.8 specifies this attribute.
- `signature-time-stamp`: this attribute is defined by the present document. It encapsulates a time-stamp token computed on the signature value for a specific signer; See clause 6.3 for details.
- `archive-time-stamp-v3`: this attribute is defined by the present document. It is used for archival of long-term signatures. More details are given in clause 6.5.2.
- `time-stamped-certs-crls-references`: this attribute is defined by the present document. It encapsulates a time-stamp token that protects a list of referenced certificates and OCSP responses and/or CRLs against certain CA compromises. See clause A.1.5.1 for details.
- `CAdES-C-time-stamp`: this attribute is defined by the present document. It is a special-purpose `TimeStampToken` for CAdES-C (clause B.1) and time-stamps the hash of the electronic signature and the complete validation data, see clause A.1.5.2 for details.

• Other signed attributes.

- `signing-time:` as defined in CMS (RFC 5652 [7]), indicates the time of the signature, as claimed by the signer. Details are provided in clause 6.2.1.
- `content-hints:` as defined in ESS (RFC 2634 [2]), provides information that describes the innermost signed content of a multi-layer message where one content is encapsulated in another. Clause 6.2.4.1 provides the specification details.
- `content-reference`: as defined in ESS (RFC 2634 [2]), can be incorporated as a way to link request and reply messages in an exchange between two parties. Clause 6.2.11 provides the specification details.
- `content-identifier`: as defined in ESS (RFC 2634 [2]), contains an identifier that may be used later on in the previous `content-reference` attribute. Clause 6.2.12 provides the specification details.
- `commitment-type-indication`: this attribute is defined by the present document. It allows the signer to indicate the commitment endorsed by the signer when producing the signature. Clause 6.2.3 provides the specification details.
- `signer-location`: this attribute is defined by the present document. It allows indicating the place where the signer purportedly produced the signature. Clause 6.2.5 provides the specification details.
- `signer-attributes` and `signer-attributes-v2`: these attributes are defined by the present document The allow incorporating signer attributes (e.g. signer roles). The first one, `signer-attributes`, specified in clause 6.2.6.1 allows incorporating claimed or certified attributes. The certified attributes are included using X509 based attribute certificates, issued by an Attribute Authority. The `signer-attributes-v2` attribute, which is specified in clause 6.2.6.2 allows incorporating the aforementioned types of attributes and also signed assertions and provides a possibility to add more general types of attribute certificates.
- `mime-type`: this attribute is defined by the present document. It allows the signer to indicate a mime-type. Clause 6.2.4.2 provides the specification details.
- `signature-policy-store`: this attribute is defined by the present document. It allows incorporating to the signature the signature policy document or a pointer to a local storage where this document may be stored for managing the signature in the long term.. Clause 6.2.10 provides the specification details.

• Other unsigned attributes.

- `CounterSignature`, as defined in CMS (RFC 5652 [7]); it can be incorporated wherever counter-signatures (i.e. a signature on a previous signature) are needed. Clause 6.2.7 provides the specification details.

The previous list of attributes can be combined to generate different electronic signature forms. Clause 4.1 specifies signatures forms that are further profiled in CAdES Baseline Profile (part 2 of the present document). Additional extended forms are defined in the normative annex B.

4.1 Electronic Signature formats

The current clause defines four forms of CMS-based advanced electronic signatures (CAdES), namely, the **Basic Electronic Signature** (CAdES-BES), **Explicit Policy-based Electronic Signature** (CAdES-EPES), the **Electronic Signature with Time** (CAdES-T) and the **Archival Electronic Signature** (CAdES-A).

The normative annex B defines forms of CAdES signatures using attributes that contain references to validation data and attributes that encapsulate time-stamp tokens on the aforementioned references.

4.1.1 CAdES Basic Electronic Signature (CAdES-BES)

A CAdES **Basic Electronic Signature** (CAdES-BES), in accordance with the present document contains:

- the signed user data (e.g. the signer's document), as defined in CMS (RFC 5652 [7]);
- a collection of mandatory signed attributes; and
- the digital signature value computed on the user data and, when present, on the signed attributes, as defined in CMS (RFC 5652 [7]).

This form represents an AdES-BES signature as described in ETSI EN 319 102 [i.5].

A CAdES-BES may contain a collection of signed and unsigned attributes.

The mandatory signed attributes are:

- `content-type`
- `message-digest`
- `ESS signing-certificate` OR `ESS signing-certificate-v2`. A CAdES-BES claiming compliance with the present document shall include one of them.

Optional signed attributes may be added to the CAdES-BES, including optional signed attributes defined in CMS (RFC 5652 [7]), ESS (RFC 2634 [2]), and the present document:

- `signing-time`.
- `content-hints`
- `content-reference`
- `content-identifier`
- `commitment-type-indication`
- `signer-location`.
- `signer-attributes` or `signer-attributes-v2`.
- `content-time-stamp`
- `mime-type`.

A CAdES-BES form can also incorporate instances of unsigned attributes, as defined in CMS (RFC 5652 [7]) and ESS (RFC 2634 [2]).

- `countersignature`

The structure of the CAdES-BES is illustrated in figure 1.

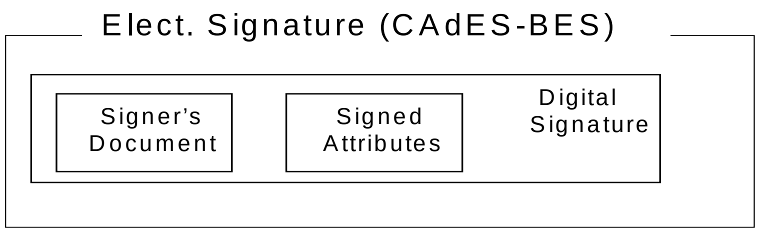


Figure 1: Illustration of a CAdES-BES

The signer's conformance requirements of a CAdES-BES are defined in clause 7.1.

NOTE: The CAdES-BES is the minimum form 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.

The CAdES-BES satisfies the legal requirements for electronic signatures, as defined in the European Directive on Electronic Signatures [i.16]. It provides authentication and integrity protection.

A counter-signature on a CAdES-BES will be created after the signature and added encapsulated in the unsigned `countersignature` attribute. The counter-signature covers the original CAdES-BES signature.

The semantics of the signed data of a CAdES-BES or its context may implicitly indicate a signature policy to the verifier. Specification of the contents of signature policies is outside the scope of the present document. However, further information on signature policies is provided in EN 319 272-1 [i.8].

4.1.2 CAdES Explicit Policy-based Electronic Signatures (CAdES-EPES)

A CAdES **Explicit Policy-based Electronic Signature** (CAdES-EPES), in accordance with the present document, extends the definition of an electronic signature to conform to the identified signature policy. A CAdES-EPES incorporates a signed attribute (`sigPolicyID` attribute) indicating the signature policy that shall be used to validate the electronic signature. This signed attribute is protected by the signature. The signature may also have other signed attributes required to conform to the mandated signature policy.

This form represents an AdES-EPES signature as described in ETSI EN 319 102 [i.5].

Clause 6.2.9 provides the details on the specification of `signature-policy-identifier` attribute

Further information on signature policies is provided in EN 319 172-1 [i.8].

The structure of the CAdES-EPES is illustrated in figure 2.

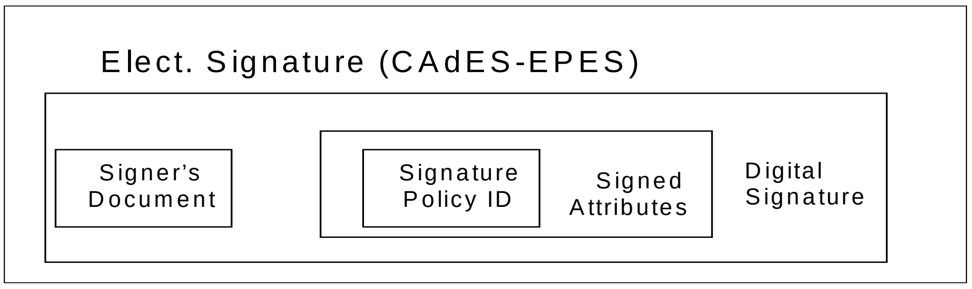


Figure 2: Illustration of a CAdES-EPES

The signer's conformance requirements of CAdES-EPES are defined in clause 7.2.

A counter-signature on a CAdES-EPES will be created after the signature and added encapsulated in the unsigned `countersignature` attribute. The counter-signature covers the original CAdES-EPES signature.

4.1.3 Electronic Signature formats with validation data

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

- Public Key Certificates (PKCs);
- revocation status information for each PKC;
- trusted time-stamps applied to the digital signature, otherwise a time-mark 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` signed attribute is present, the validation data 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 online 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 unsigned attributes able to contain validation data that can be added to CAdES-BES and CAdES-EPES, leading to electronic signature formats that include validation data. Clauses 4.1.3.1 and 4.1.3.2 describes the basic formats including validation data.

4.1.3.1 Electronic Signature with time (CAdES-T)

A CAdES **Electronic Signature with Time** (CAdES-T), in accordance with the present document, is an electronic signature for which a Trust Service Provider has generated a trusted token proving that the signature itself actually existed at a certain date and time.

The trusted time may be provided by two different means:

- a time-stamp token encapsulated by the `signature-time-stamp` attribute as an unsigned attribute added to the electronic signature; or
- a time-mark of the electronic signature provided by a Trust Service Provider.

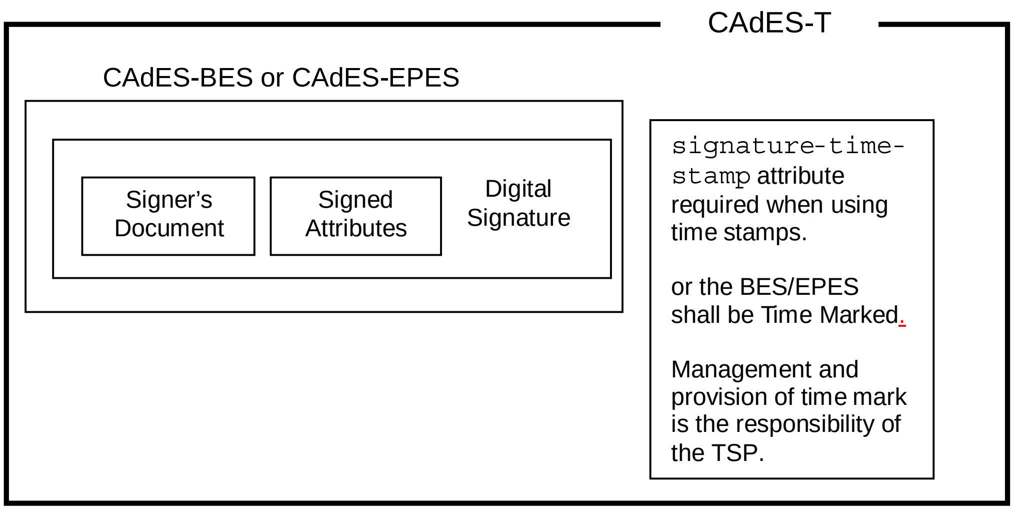
This form represents an AdES-T signature as described in ETSI EN 319 102 [i.5].

The `signature-time-stamp` attribute contains a time-stamp token of the electronic signature value. Clause 6.3 provides the specification details.

A time-mark provided by a Trust Service would have a similar effect as the `signature-time-stamp` attribute, but in this case, no attribute 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 present document.

EDITOR NOTE: Would it be useful to include a referencing mechanism to the time-mark? We are especially interested in the opinion of stakeholders using a time-mark mechanism.

Trusted time provides the initial steps towards providing long-term validity. Electronic signatures with the `signature-time-stamp` attribute or a time-marked BES/EPES forming the CAdES-T are illustrated in figure 3.


CAdES-T
CAdES-BES or CAdES-EPES
Signer's Document
Signed Attributes
Digital Signature
signature-time-stamp attribute required when using time stamps.
or the BES/EPES shall be Time Marked.
Management and provision of time mark is the responsibility of the TSP.

NOTE 1: A time-stamp token is added to the CAdES-BES or CAdES-EPES as an unsigned attribute.

Figure 3: Illustration of CAdES-T formats

4.1.3.2 CAdES-A with archive-time-stamp (ATSv3) attribute

A CAdES **Archival Electronic Signature** (CAdES-A) with `archive-time-stamp-v3` attribute, in accordance with the present document may be build on CAdES-BES, CAdES-EPES or any format above, by adding one or more `archive-time-stamp-v3` attributes. This form is used for archival of long-term signatures. Successive time-stamps protect the whole material against vulnerable hashing algorithms or the breaking of the cryptographic material or algorithms. Clause 6.5 contains the specification details.

NOTE 1: It is possible to create CAdES-A starting from CAdES-BES or CAdES-EPES. The archive time-stamp gives an assurance when the signature existed already. However, it is highly recommended to build CAdES-A upon CAdES-T, especially if the archiving is done long after the creation of the signature.

EDITOR NOTE: In TS 101 733 v2.2.1 it was stated that CAdES-A can be build on CAdES-BES and CAdES-EPES. We recommend to build it on CAdES-T but for the moment we keep the possibility to build it on CAdES-BES/EPES for backward compatible. Feedback from stakeholders is kindly requested on this topic.

This form represents an AdES-LT signature as described in ETSI EN 319 102 [i.5].

NOTE 2: This form of electronic signature also provides protection against a TSP key compromise.

The structure of the CAdES-A with ATSv3 form build on CAdES-T is shown in figure 4.

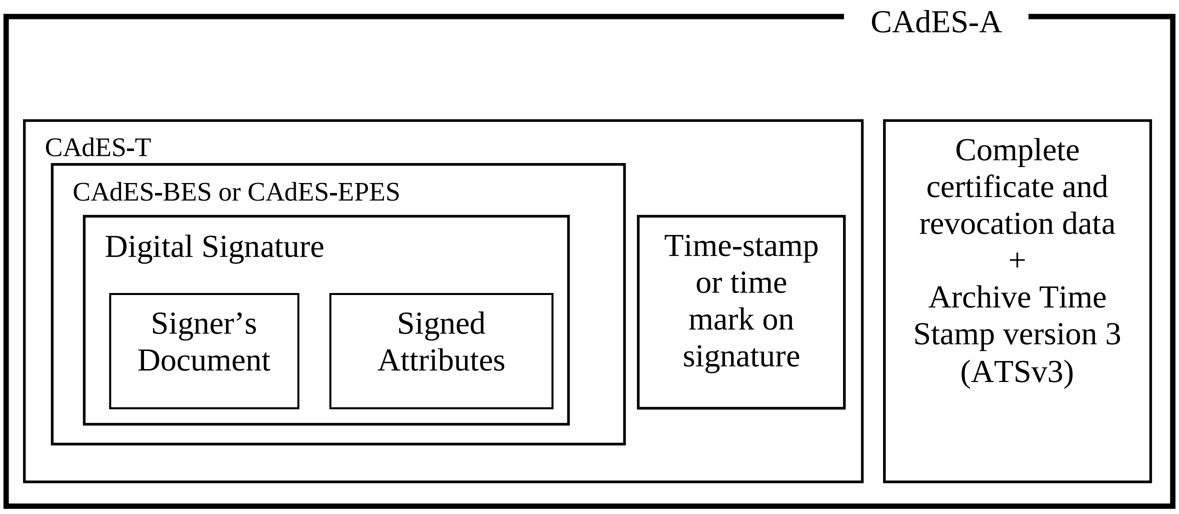

CAdES-A
CAdES-T
CAdES-BES or CAdES-EPES
Digital Signature
Signer's Document
Signed Attributes
Time-stamp or time mark on signature
Complete certificate and revocation data + Archive Time Stamp version 3 (ATSv3)

Figure 4: Illustration of CAdES-A

5 General syntax

CAdES signatures build on Cryptographic Message Syntax (CMS), as defined in RFC 5652 [7], by incorporation of signed and unsigned attributes, which are specified in RFC 5652[7], RFC 2634 [2], and the present document.

NOTE: CMS defines content types for `id-data`, `id-signedData`, `id-envelopedData`, `id-digestedData`, `id-encryptedData`, and `id-authenticatedData`. Although CMS permits other documents to define other content types, it states that the ASN.1 type defined should not be a CHOICE type. The present document does not define other content types.

The following sub-clauses list the types that are used in the attributes described in Clause 6.

5.1 The data content type

The data content type of the Electronic Signatures is as defined in CMS (RFC 5652 [7]).

NOTE: If the content type is `id-data`, it is recommended that the content be encoded using MIME, and that the MIME type is used to identify the presentation format of the data. See clause F.1 for an example of using MIME to identify the encoding type.

5.2 The `signed-data` content type

The `signed-data` content type of the Electronic Signatures is as defined in CMS (RFC 5652 [7]).

5.3 The `SignedData` type

The syntax of the `SignedData` of the Electronic Signatures is as defined in CMS (RFC 5652 [7]).

The fields of type `SignedData` are as defined in CMS (RFC 5652 [7]).

The degenerate case, where there are no signers, is not valid in the present document.

5.4 The `EncapsulatedContentInfo` type

The syntax of the `EncapsulatedContentInfo` type of the Electronic Signature is as defined in CMS (RFC 5652 [7]).

For the purpose of long-term validation, as defined by the present document, it is advisable that either the `eContent` is present, or the data that is signed is archived in such as way as to preserve any data encoding. It is important that the OCTET STRING used to generate the signature remains the same every time either the verifier or an arbitrator validates the signature.

NOTE: The `eContent` is optional in CMS:

- When it is present, this allows the signed data to be encapsulated in the `SignedData structure` which then contains both the signed data and the signature. However, the signed data may only be accessed by a verifier able to decode the ASN.1 encoded `SignedData` structure.
- When it is missing, this allows the signed data to be sent or stored separately from the signature, and the SignedData structure only contains the signature. Under these circumstances, the data object that is signed needs to be stored and distributed in such a way as to preserve any data encoding

The degenerate case where there are no signers is not valid in the present document.

5.5 The `SignerInfo` type

The syntax of the `SignerInfo` type of the Electronic Signature is as defined in CMS (RFC 5652 [7]).

Per-signer information is represented in the type `SignerInfo`. In the case of multiple parallel signatures, there is one instance of this field for each signer.

The fields of type `SignerInfo` have the meanings defined in CMS (RFC 5652 [7]). For each of them the signedAttrs field shall be present and shall contain the following attributes:

- `content-type`, as defined in clause 6.1.1;
- `message-digest`, as defined in clause 6.1.2; and
- `signing-certificate`, as defined in clause 6.2.2

5.6 Other Standard Data Structures

5.6.1 Time-Stamp Token Format

The `TimeStampToken` type is defined in RFC 3161 [3]. Time-stamp tokens are profiled in TS 101 861 [i.2].

Implementations of the present document shall support the usage of both the `signing-certificate` attribute and the `signing-certificate-v2` attribute, within time-stamp tokens, in accordance with RFC 5035 [5].

5.6.2 Name and Attribute Formats

.The present document does not specify the format of the signer's attribute that may be included in public key certificates.

NOTE 1: The name used by the signer, held as the subject in the signer's certificate, is allocated and verified on registration with the Certification Authority, see the policy requirements for CAs issuing qualified certificates (ETSI EN 319 411-2 [i.11]) or public key certificates (ETSI EN 319 411-3 [i.12]) for more details. The format of the certificates for natural persons and legal persons is specified in the profiles ETSI EN 319 412-2 [i.14] and ETSI EN 319 412-3 [i.15], respectively.

NOTE 2: The signer's attribute may be supported by using a *claimed* role in the CMS signed attributes field or by placing an `AttributeCertificate`, as defined in RFC 5755 [8], containing a *certified* role in the CMS signed attributes field;

6 Attribute syntax

This clause provides details on attributes specified within CMS (RFC 5652 [7]), ESS (RFC 2634 [2]), and specifies new attributes for building CAdES signatures. The overall structure of an Electronic Signature is as defined in CMS. The Electronic Signature uses attributes defined in CMS, ESS, and the present document. The present document defines Electronic Signature attributes that it uses and that are not defined elsewhere.

6.1 CMS defined mandatory signed attributes

The attributes in this clause are mandatory signed attributes as defined in CMS (RFC 5652 [7]).

6.1.1 The `content-type` attribute

The `content-type` attribute is a mandatory signed attribute. It indicates the type of the signed content. The syntax of the `content-type` attribute type is as defined in CMS (RFC 5652 [7], clause 11.1).

NOTE 1: CMS defines content types for `id-data`, `id-signedData`, `id-envelopedData`, `id-digestedData`, `id-encryptedData`, and `id-authenticatedData`. Although CMS permits other documents to define other content types, the ASN.1 type defined should not be a CHOICE type. The present document does not define other content types.

NOTE 2: As stated in RFC 5652 [7], the content of ContentType (the value of the attribute `content-type`) is the same as the eContentType of the EncapsulatedContentInfo value being signed.

NOTE 3: For implementations supporting signature generation, if the content-type attribute is `id-data`, then it is recommended that the `eContent` be encoded using MIME. For implementations supporting signature verification, if the signed data (i.e. `eContent`) is MIME-encoded, then the OID of the `content-type` attribute is `id-data`. In both cases, the MIME `Content-Type` is used to identify the format of the data such that an appropriate presentation mechanism can be chosen. See annex F for further details about the use of MIME.

6.1.2 The `message-digest` attribute

The `message-digest` attributed is a mandatory signed attribute. The syntax of the `message-digest` attribute type of the Electronic Signature is as defined in CMS (RFC 5652 [7]).

The message digest calculation process is as defined in CMS (RFC 5652 [7]).

6.2 Attributes for CAdES-BES and CAdES-EPES forms

6.2.1 The `signing-time` attribute

The `signing-time` attribute is an optional signed attribute. The `signing-time` attribute specifies the time at which the signer claims to have performed the signing process.

`Signing-time` attribute values for Electronic Signature have the ASN.1 type `SigningTime` as defined in CMS (RFC 5652 [7]).

NOTE: RFC 5652 [7] states that "dates between January 1, 1950 and December 31, 2049 (inclusive) must be encoded as UTCTime. Any dates with year values before 1950 or after 2049 must be encoded as GeneralizedTime".

6.2.2 Signing certificate reference attributes

The Signing certificate reference attributes are supported by using either the ESS `signing-certificate` attribute or the `ESS signing-certificate-v2` attribute. The signing certificate reference attribute shall be present in the signed-data defined by the present document

Any CAdES signature shall contain one, and only one, of the following signed attributes with the `signedData`:

- The ESS `signing-certificate` attribute, defined in ESS RFC 2634 [2], shall be used if the SHA-1 hashing algorithm is used.
- The ESS `signing-certificate-v2` attribute, defined in "ESS Update: Adding CertID Algorithm Agility", RFC 5035 [5], which shall be used when other hashing algorithms are to be used.

Signed attributes ESS `signing-certificate` and `ESS signing-certificate-v2` encapsulate references to the signing certificate and optionally to other certificates in the certification path.

These attributes are designed to prevent simple substitution and reissue attacks and to allow for a restricted set of certificates to be used in verifying a signature. They have a compact form (much shorter than the full certificate) that allows for a certificate to be unambiguously identified.

The certificate to be used to verify the signature (i.e. the certificate from the signer) shall be present in the signing certificate reference attribute. The signature validation policy may mandate other certificates be present that may include all the certificates up to the trust anchor.

NOTE: RFC 2634 and RFC 5035 state that the first certificate in the sequence is the certificate used to verify the signature.

6.2.2.1 ESS `signing-certificate` attribute

The ESS `signing-certificate` attribute shall be a signed attribute. The syntax of the `signing-certificate` attribute type of the Electronic Signature is as defined in Enhanced Security Services (ESS), RFC 2634 [2], and further specified in the present document.

The `policies` field is not used in the present document.

The encoding of the `ESSCertID` for this certificate shall include the `issuerSerial` field.

If present, the `issuerAndSerialNumber` in `SignerIdentifier` field of the `SignerInfo` shall match the `issuerSerial` field present in `ESSCertID`. In addition, the `certHash` from `ESSCertID` shall match the SHA-1 hash of the certificate.

NOTE 1: Where an attribute certificate is used by the signer to associate a role, or other attributes of the signer, within the electronic signature, this is placed in the `signer-attributes` attribute as defined in clause 6.2.6.

NOTE 2: The use of SHA-1 is being phased out in some countries and hence the use of the signing-certificate-v2 attribute is recommended.

6.2.2.2 ESS `signing-certificate-v2` attribute

The ESS `signing-certificate-v2` attribute shall be a signed attribute.

The ESS `signing-certificate-v2` attribute is similar to the ESS `signing-certificate` defined above, except that this attribute can be used with hashing algorithms other than SHA-1.

The syntax of the `signing-certificate-v2` attribute type of the Electronic Signature is as defined in "ESS Update: Adding CertID Algorithm Agility", RFC 5035 [5], and further specified in the present document.

The `policies` field is not used in the present document.

This attribute shall be used in the same manner as defined above for the ESS `signing-certificate` attribute.

The object identifier for this attribute is:

```
      id-aa-signingCertificateV2 OBJECT IDENTIFIER ::= { iso(1)
          member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
          smime(16) id-aa(2) 47 }
```

If present, the `issuerAndSerialNumber` in `SignerIdentifier` field of the `SignerInfo` shall match the `issuerSerial` field present in `ESSCertIDv2`. In addition, the `certHash` from `ESSCertIDv2` shall match the hash of the certificate computed using the hash function specified in the `hashAlgorithm` field. Implementations of the present document shall support the usage of both the `signing-certificate` attribute and the `signing-certificate-v2` attribute, within time-stamp tokens, in accordance with RFC 5035 [5].

NOTE 1: Where an attribute certificate is used by the signer to associate a role, or other attributes of the signer, within the electronic signature, this is placed in the `signer-attributes` attribute as defined in clause 6.2.6.

NOTE 2: Previous versions of TS 101 733 used the other signing certificate attribute (see clause A.2.1) for the same purpose. Its use is now deprecated, since this structure is simpler.

6.2.2.3 Use of signing certificate reference attributes in the validation

The certificate identified by the signing certificate reference attribute (`signing-certificate` or `signing-certificate-v2`) shall be used during the signature verification process. If the hash of the certificate does not match the certificate used to verify the signature, the signature shall be considered invalid.

6.2.3 The `commitment-type-indication` attribute

The `commitment-type-indication` attribute is an optional signed attribute defined in the present document. At most one single instance of this attribute shall be present. If the attribute is present, it shall contain exactly one component of `AttributeValue` type.

The following object identifier identifies the `commitment-type-indication` attribute:

```
id-aa-ets-commitmentType OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 16}
```

`commitment-type-indication` attribute values have ASN.1 type `CommitmentTypeIndication.`

```
CommitmentTypeIndication ::= SEQUENCE {
  commitmentTypeId CommitmentTypeIdentifier,
  commitmentTypeQualifier SEQUENCE SIZE (1..MAX) OF CommitmentTypeQualifier OPTIONAL}

CommitmentTypeIdentifier ::= OBJECT IDENTIFIER

CommitmentTypeQualifier ::= SEQUENCE {
        commitmentTypeIdentifier CommitmentTypeIdentifier,
        qualifier   ANY DEFINED BY commitmentTypeIdentifier}
```

There may be situations where a signer wants to explicitly indicate to a verifier that by signing the data, it illustrates a type of commitment on behalf of the signer. The `commitment-type-indication` attribute conveys such information. The commitment type may be:

- defined as part of the signature policy, in which case, the commitment type has precise semantics that are defined as part of the signature policy; or
- be 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 specification of commitment type qualifiers is outside the scope of the present document.

The following generic commitment types are defined in the present document:

```
    id-cti-ets-proofOfOrigin OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 1}
```

```
id-cti-ets-proofOfReceipt OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 2}

id-cti-ets-proofOfDelivery OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 3}

id-cti-ets-proofOfSender OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 4}

id-cti-ets-proofOfApproval OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 5}

id-cti-ets-proofOfCreation OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 6}
```

These generic commitment types have the following meanings:

- **Proof of origin** indicates that the signer recognizes to have created, approved, and sent the message.
- **Proof of receipt** indicates that signer recognizes to have received the content of the message.
- **Proof of delivery** indicates that the TSP providing that indication has delivered a message in a local store accessible to the recipient of the message.
- **Proof of sender** indicates that the entity providing that indication has sent the message (but not necessarily created it).
- **Proof of approval** indicates that the signer has approved the content of the message.
- **Proof of creation** indicates that the signer has created the message (but not necessarily approved, nor sent it).

6.2.4 Attributes for identifying the signed data type

6.2.4.1 The content-hints attribute

The syntax of the content-hints attribute type of the Electronic Signature is as defined in ESS (RFC 2634 [2]).

The content-hints attribute provides information on the innermost signed content of a multi-layer message where one content is encapsulated in another. It is an optional attribute

When used to indicate the precise format of the data to be presented to the user, the following rules apply:

- the contentType indicates the type of the associated content. It is an object identifier assigned by an authority that defines the content type; and
- when the contentType is id-data the contentDescription shall define the presentation format; the format may be defined by MIME types.

When the format of the content is defined by MIME types, the following rules apply:

- the contentType shall be id-data as defined in CMS (RFC 5652 [7]);
- the contentDescription shall be used to indicate the encoding and the intended presentation application of the data, in accordance with the rules defined RFC 2045 [1]; see annex F for an example of structured contents and MIME.

NOTE 1: id-data OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 1 }.

NOTE 2: contentDescription is optional in ESS (RFC 2634 [2]). It may be used to complement a contentType defined elsewhere; such additional information is outside the scope of the present document.

6.2.4.2 The `mime-type` attribute

The `mime-type` attribute is an optional signed attribute defined in the present document. Only a single `mime-type` attribute shall be present.

The `mime-type` attribute shall not be used within a countersignature.

The following object identifier identifies the `mime-type` attribute:

```
id-aa-ets-mimeType OBJECT IDENTIFIER ::= { itu-t(0) identified-organization(4) etsi(0) electronic-
signature-standard (1733) attributes(2) 1 }
```

`mime-type` attribute values have ASN.1 type UTF8String:

```
mimeType::= UTF8String
```

The `mime-type` attribute is an attribute that lets the signature generator indicate the mime-type of the signed data. It is similar in spirit to the `contentDescription` field of the `content-hints` attribute, but can be used without a multi-layered document.

The `mimeType` is used to indicate the encoding and the intended presentation application of the signed data, in accordance with the rules defined in RFC 2045 [1]; see annex F for an example of structured contents and MIME.

6.2.5 The `signer-location` attribute

The `signer-location` attribute is an optional signed attribute defined in the present document. At most one single instance of this attribute shall be present. If the attribute is present, it shall contain exactly one component of `AttributeValue` type.

The `signer-location` attribute specifies an address associated with the signer at a particular geographical (e.g. city) location. The address is registered in the country in which the signer is located.

The following object identifier identifies the signer-location attribute:

```
id-aa-ets-signerLocation OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 17}
```

Signer-location attribute values have ASN.1 type `SignerLocation`:

```
SignerLocation ::= SEQUENCE { -- at least one of the following shall be present:
            countryName [0] DirectoryString OPTIONAL,
                -- As used to name a Country in X.500
            localityName [1] DirectoryString OPTIONAL,
                -- As used to name a locality in X.500
            postalAdddress [2] PostalAddress OPTIONAL }

PostalAddress ::= SEQUENCE SIZE(1..6) OF DirectoryString
```

6.2.6 Incorporating attributes of the signer

The present document specifies two optional signed attributes for incorporating attributes of the signer namlely:

- `signer-attributes` (see clause 6.2.6.1
- `signer-attributes-v2` (see clause 6.2.6.2)

At most one single instance of a signer attributes attribute may be present within a CAdES signature, i.e. either a single instance of the `signer-attributes` attribute or a single instance of the `signers-attributes-v2` attribute. . If the attribute is present, it shall contain exactly one `AttributeValue`.

The signer attributes allows incorporating to the signature additional attributes of the signer (e.g. role). The present document differentiates three types of attributes:

- claimed attributes of the signer;
- certified attributes of the signer; or
- assertions signed by a third party (only in `signer-attributes-v2`).

Both attributes, `signer-attributes` and `signer-attributes-v2`, allow to define a claimed attribute using an `Attribute` and to define a certified attribute using an `AttributeCertificate`.

Clause 6.2.6.3 defines a new attribute that can be used to describe a claimed role by encapsulating a SAML token.

NOTE 1 `Attribute` and `AttributeCertificate` are as defined, respectively, in Recommendation ITU-T X.501 [13] and ITU-T X.509 [14].

NOTE 2 A user who wants to add a claimed role to one of the signer-attributes attributes may use the `RoleAttribute` as defined in X.509 [14]

6.2.6.1 The `signer-attributes` attribute

The `signer-attributes` attribute is an optional signed attribute defined in the present document. The following object identifier identifies the signer-attribute attribute:

```
id-aa-ets-signerAttr OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 18}
```

`signer-attributes` values have ASN.1 type `SignerAttribute`:

```
SignerAttribute ::= SEQUENCE OF CHOICE {
            claimedAttributes   [0] ClaimedAttributes,
            certifiedAttributes [1] CertifiedAttributes }

ClaimedAttributes ::= SEQUENCE OF Attribute

CertifiedAttributes ::= AttributeCertificate  -- as defined in RFC 3281: see clause 4.1.
```

The `claimedAttributes` field contains a sequence of attributes claimed by the signer but not certified. These signer attributes are expressed using `Attribute` types.

The `certifiedAttributes` field contains a sequence of certified attributes, expressed by the inclusion of attribute certificates.

6.2.6.2 The `signer-attributes-v2` attribute

The `signer-attributes-v2` attribute is an optional signed attribute defined in the present document.

The following object identifier identifies the signer-attribute attribute:

```
id-aa-ets-signerAttrV2 OBJECT IDENTIFIER ::= xxxx
```

`signer-attributes-v2 values have ASN.1 type SignerAttributeV2:`

```
SignerAttributeV2 ::= SEQUENCE {
            claimedAttributes       [0] ClaimedAttributes OPTIONAL,
            certifiedAttributesV2   [1] CertifiedAttributesV2 OPTIONAL,
            signedAssertions        [2] SignedAssertions OPTIONAL}

ClaimedAttributes ::= SEQUENCE OF Attribute

CertifiedAttributesV2 ::= SEQUENCE OF CHOICE {
            attributeCertificate        [0] AttributeCertificate,
                                            -- as defined in RFC 3281: see clause 4.1.
            otherAttributeCertificate   [1] OtherAttributeCertificate }

OtherAttributeCertificate ::= SEQUENCE {
        otherAttributeCertID        OtherAttributeCertID,
        otherAttributeCert          ANY DEFINED BY otherAttributeCertID }
```

```
OtherAttributeCertID ::= OBJECT IDENTIFIER

SignedAssertions ::= SEQUENCE OF SignedAssertion

SignedAssertion ::= SEQUENCE {
        signedAssertionID       SignedAssertionID,
        signedAssertion         ANY DEFINED BY signedAssertionID }

SignedAssertionID ::= OBJECT IDENTIFIER
```

The claimedAttributes field contains a sequence of attributes claimed by the signer but which are not certified. These signer attributes are expressed using Attribute types.

The certifiedAttributes field contains a sequence of certified attributes. These signer attributes can be expressed by:

- attributeCertificate: an attribute certificates delivered by AAs
- otherAttributeCertificate: an attribute certificates (issued, in consequence, by Attribute Authorities) in different syntax than the one used for X509 attribute certificates. The definition of specific

otherAttributeCertificates is outside of the scope of the present document. Any attribute certificate encapsulated within this sequence shall be signed by a trusted attribute authority.

NOTE 1: EN 319 411-4 [i.13], annex A specifies general requirements for attribute certificates.

The signedAssertions field contains a sequence of assertions signed by a third party. The definition of specific signedAssertions is outside of the scope of the present document. Any assertion encapsulated within this sequence shall be signed by third party.

NOTE 2: A signed assertion is stronger than a claimed attribute, since a third party asserts with a signature that the attribute of the signer is valid. However, it is less restrictive than an attribute certificate.

NOTE 3: A possible content of such a qualifier might be a signed SAML token, see of SAML v1.1 [i.28] and SAML v2.0 [i.29]

The signer-attributes-v2 attribute shall contain at least one attribute.

6.2.6.3 The claimed-SAML-assertion attribute

The claimed-SAML-assertion attribute shall be used to claim a signer attribute using a SAML assertion within the claimedAttributes field of the signer-attributes or the signer-attributes-v2 attribute.

The following object identifier identifies the signer-attribute attribute:

```
id-aa-ets-claimedSAML OBJECT IDENTIFIER ::= xxxx
```

claimed-SAML-assertion values have ASN.1 type ClaimedSAMLAssertion

```
ClaimedSAMLAssertion ::= OCTET STRING.
```

The claimed-SAML-assertion attribute shall contain the byte representation of SAML assertion.

NOTE: For more details on SAML tokens see the definitions of SAML v1.1 [i.28] and SAML v2.0 [i.29]

6.2.7 The countersignature attribute

The countersignature attribute is an optional unsigned attribute.

The countersignature attribute values for Electronic Signature have ASN.1 type CounterSignature, as defined in CMS (RFC 5652 [7]). It is an optional attribute.

6.2.8 The `content-time-stamp` attribute

The `content-time-stamp` attribute is an optional signed attribute defined in the present document. Several instances of this attribute may occur with an electronic signature, from different TSAs.

The following object identifier identifies the `content-time-stamp` attribute:

```
id-aa-ets-contentTimestamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 20}
```

`content-time-stamp` attribute values have ASN.1 type ContentTimestamp:

```
ContentTimestamp::= TimeStampToken
```

The value of `messageImprint` of `TimeStampToken` (as described in RFC 3161 [3]) shall be a hash of

- the value of `eContent` in the case of an attached signatures, or
- the external data in the case of a detached signature.

In both cases, the hash is computed over the raw data, without ASN.1 tag and length.

The `content-time-stamp` attribute is an attribute that encapsulates a time-stamp token of the signed data content before it is signed.

For further information and definition of `TimeStampToken,` see clause 5.6.1.

NOTE: `content-time-stamp` indicates that the signed information was formed before the date and time included in the `content-time-stamp`.

6.2.9 The `signature-policy-identifier` attribute (CAdES-EPES)

The `signature-policy-identifier` attribute is an optional signed attribute defined in the present document. At most one single instance of this attribute shall be present. If the attribute is present, it shall contain exactly one component of `AttributeValue` type.

A signature policy defines the rules for creation and validation of an electronic signature. The present document mandates that for Explicit Policy-based Electronic Signatures (CAdES-EPES), the `signature-policy-identifier` attribute shall be present and shall contain an explicit reference to the signature policy.

```
The following object identifier identifies the signature-policy-identifier attribute:
id-aa-ets-sigPolicyId OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-aa(2) 15 }

signature-policy-identifier attribute values have ASN.1 type SignaturePolicyIdentifier:
SignaturePolicyIdentifier ::=CHOICE{
        signaturePolicyId          SignaturePolicyId,
        signaturePolicyImplied     SignaturePolicyImplied -- not used in this version}

SignaturePolicyId ::= SEQUENCE {
        sigPolicyId           SigPolicyId,
        sigPolicyHash         SigPolicyHash,
        sigPolicyQualifiers   SEQUENCE SIZE (1..MAX) OF
                                SigPolicyQualifierInfo OPTIONAL}

SignaturePolicyImplied ::= NULL

SigPolicyHash ::= OtherHashAlgAndValue

OtherHashAlgAndValue ::= SEQUENCE {
      hashAlgorithm   AlgorithmIdentifier,
      hashValue       OtherHashValue }

OtherHashValue ::= OCTET STRING

SigPolicyId ::= OBJECT IDENTIFIER
```

The `sigPolicyId` field contains an object-identifier that uniquely identifies a specific version of the signature policy.

The `sigPolicyHash` field contains the identifier of the hash algorithm and the hash of the value of the signature policy. The `hashValue` within the `sigPolicyHash` may be set to zero to indicate that the policy hash value is not known.

NOTE: The use of a zero-`sigPolicyHash` value is to ensure backwards compatibility with earlier versions of TS 101 733. If `sigPolicyHash` is zero, then the hash value should not be checked against the calculated hash value of the signature policy.

The input to hash computation of `sigPolicyHash` depends on the technical specification of the signature policy. In the case where the specification is not clear from the context of the signature, the sp-doc-specification qualifier shall be used to identify the used specification-

6.2.9.1 SignaturePolicyQualifier

The `signature-policy-identifier` attribute may further be qualified with additional information present within `sigPolicyQualifiers.` The semantics and syntax of the qualifier is as identified by the object-identifier in the `sigPolicyQualifierId` field. The general syntax of this qualifier is as follows:

```
SigPolicyQualifierInfo ::= SEQUENCE {
        sigPolicyQualifierId  SigPolicyQualifierId,
        sigQualifier          ANY DEFINED BY sigPolicyQualifierId }
```

The present document specifies the following qualifiers:

- `spuri`: this contains the URI or URL reference to the signature policy; and
- `sp-user-notice`: this contains a user notice that should be displayed whenever the signature is validated.
- `sp-doc-specification`: an identifier of the technical specification that defines the syntax used for producing the signature policy document (`SPDocSpecification` element).

```
-- sigpolicyQualifierIds defined in the present document

SigPolicyQualifierId ::= OBJECT IDENTIFIER

    id-spq-ets-uri OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-spq(5) 1 }

SPuri ::= IA5String

    id-spq-ets-unotice OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-spq(5) 2 }

SPUserNotice ::= SEQUENCE {
        noticeRef        NoticeReference OPTIONAL,
        explicitText     DisplayText OPTIONAL}

NoticeReference ::= SEQUENCE {
        organization     DisplayText,
        noticeNumbers    SEQUENCE OF INTEGER }

DisplayText ::= CHOICE {
        visibleString    VisibleString  (SIZE (1..200)),
        bmpString        BMPString      (SIZE (1..200)),
        utf8String       UTF8String     (SIZE (1..200)) }

SPDocSpecification ::= CHOICE {
        oid     OBJECT IDENTIFIER,
        uri     IA5String }

    id-spq-ets-docspec OBJECT IDENTIFIER ::=  xxxx
```

The `SPDocSpecification` identifies the technical specification that defines the syntax of the signature policy. If the technical specification is identified using an OID, then the `oid` choice shall be used to contain the OID of the specification. If the technical specification is identified using a URI, then the uri choice shall be used to contain this URI.

EDITOR NOTE: This new qualifier will allow to identify whether the signature policy document is human readable, XML encoded, or ASN.1 encoded, by identifying the specific Technical Specifications where these formats will be defined.

6.2.10 The signature-policy-store attribute

The signature-policy-store is an optional unsigned attribute. At most one single instance of this attribute shall be present. If the attribute is present, it shall contain exactly one component of AttributeValue type.

It can be used to store the signature policy document which is referenced in the signature-policy-identifier, such that it can be used for offline and long-term validation.

The following object identifier identifies the signature-policy-store attribute:

```
id-aa-ets-sigPolicyStore OBJECT IDENTIFIER ::= xxxx
```

signature-policy-store attribute values have ASN.1 type SignaturePolicyStore:

```
SignaturePolicyStore ::=SEQUENCE {
        spDocSpec       SPDocSpecification ,
        spDocument      SignaturePolicyDocument }

SignaturePolicyDocument ::= CHOICE {
        sigPolicyEncoded        OCTET STRING,
        sigPolicyLocalURI       IA5String }
```

The spDocument may contain the encoded signature policy document as content of the sigPolicyEncoded element, or an URI to a local store where this document may be retrieved as sigPolicyLocalURI.

NOTE 1: The URI in sigPolicyLocalURI may be different than the SPuri qualifier's value.

The spDocSpec identifies the technical specification that defines the syntax of the signature policy

NOTE 2: It is the responsibility of the entity adding the signature policy into the signature-policy-store to make sure that the correct document is stored.

6.2.11 The content-reference attribute

The content-reference attribute is an optional attributed defined in ESS (RFC 2634 [2]). It shall be a signed attribute.

The content-reference attribute is a link from one SignedData to another. It may be used to link a reply to the original message to which it refers, or to incorporate by reference one SignedData into another.

content-reference attribute values for Electronic Signature have ASN.1 type ContentReference, as defined in ESS (RFC 2634 [2]).

The content-reference attribute shall be used as defined in ESS (RFC 2634 [2]).

6.2.12 The content-identifier attribute

The content-identifier attribute is an optional attributed defined in ESS (RFC 2634 [2]). It shall be a signed attribute.

The content-identifier attribute provides an identifier for the signed content, for use when a reference may be later required to that content; for example, in the content-reference attribute in other signed data sent later.

content-identifier attribute type values for the Electronic Signature have an ASN.1 type ContentIdentifier, as defined in ESS (RFC 2634 [2]).

The minimal content-identifier attribute should contain a concatenation of user-specific identification information (such as a user name or public keying material identification information), a GeneralizedTime string, and a random number.

6.3 The signature-time-stamp attribute (CAdES-T)

The signature-time-stamp attribute is an optional unsigned attribute. Several instances of this attribute may occur with an electronic signature, from different TSAs.

The signature-time-stamp attribute encapsulates a time-stamp token computed on the signature value for a specific signer.

The following object identifier identifies the signature-time-stamp attribute:

```
id-aa-signatureTimeStampToken OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 14}
```

The signature-time-stamp attribute value has ASN.1 type SignatureTimeStampToken:

```
SignatureTimeStampToken ::= TimeStampToken
```

The value of the messageImprint field within TimeStampToken shall be a hash of the value of the signature field (without the ASN.1 tag and length) within SignerInfo for which the signature-time-stamp attribute is created.

For further information and definition of TimeStampToken, see clause 7.4.

NOTE 1: In the case of multiple signatures, it is possible to have a:

- signature-time-stamp computed for each and all signers; or
- signature-time-stamp on some signers' signatures an none on other signers' signatures.

NOTE 2: Several time-stamp tokens, issued by different TSAs, may be present within the same signerInfo (see RFC 5652 [7]).

6.4 Validation data values

The present document specifies different places where to incorporate missing validation data before generating CAdES-A or CAdES-LT forms. See clauses 6.5 and A.1for additional details.

6.5 Archive validation data

Where an electronic signature is required to last for a very long time, and the time-stamp token on an electronic signature (signature time-stamp or previous archival time-stamps) is in danger of being invalidated due to algorithm weakness or limits in the validity period of the TSA certificate, it may be required to time-stamp the electronic signature several times. When this is required, an archive time-stamp attribute may be required for the archive form of the electronic signature (**CAdES-A**). This archive time-stamp attribute may be repeatedly applied over a period of time.

6.5.1 The ats-hash-index attribute

The ats-hash-index shall be carried as an unsigned attribute of the signature of the archive-time-stamp-v3 attribute (see clause 6.5.2). The ats-hash-index attribute shall have a single attribute value.

The following object identifier identifies the ats-hash-index unsigned attribute:

```
  id-aa-ATSHashIndex OBJECT IDENTIFIER ::= { itu-t(0) identified-organization(4)
etsi(0) electronic-signature-standard(1733) attributes(2) 5 }
```

The ats-hash-index attribute value has the ASN.1 syntax ATSHashIndex:

```
ATSHashIndex ::= SEQUENCE {
    hashIndAlgorithm AlgorithmIdentifier DEFAULT {algorithm id-sha256},
    certificatesHashIndex    SEQUENCE OF OCTET STRING,
    crlsHashIndex            SEQUENCE OF OCTET STRING,
    unsignedAttrsHashIndex   SEQUENCE OF OCTET STRING
}
```

The ats-hash-index unsigned attribute provides an unambiguous imprint of the essential components of a CAdES signature for use in the archive time-stamp (see clause 6.5.2). These essential components are elements of the following ASN.1 SET OF structures: unsignedAttrs, SignedData.certificates, and SignedData.crls.

NOTE 2: The SignedData.crls component as defined in RFC 5652 [7] can include OCSP and/or CRL revocation information.

The field hashIndAlgorithm contains an identifier of the hash algorithm used to compute the hash values contained in certificatesHashIndex, crlsHashIndex, and unsignedAttrsHashIndex. This algorithm shall be the same as the hash algorithm used for computing the archive time-stamp's message imprint.

The field certificatesHashIndex is a sequence of octet strings. Each one contains the hash value of one instance of CertificateChoices within certificates field of the root SignedData. A hash value for every instance of CertificateChoices, as present at the time when the corresponding archive time-stamp is requested, shall be included in certificatesHashIndex. No other hash value shall be included in this field.

The field crlsHashIndex is a sequence of octet strings. Each one contains the hash value of one instance of RevocationInfoChoice within crls field of the root SignedData. A hash value for every instance of RevocationInfoChoice, as present at the time when the corresponding archive time-stamp is requested, shall be included in crlsHashIndex. No other hash values shall be included in this field.

The field unsignedAttrsHashIndex is a sequence of octet strings. Each one contains the hash value of one instance of Attribute within unsignedAttrs field of the SignerInfo. A hash value for every instance of Attribute, as present at the time when the corresponding archive time-stamp is requested, shall be included in unsignedAttrsHashIndex. No other hash values shall be included in this field.

Each hash shall be the result, encapsulated in an octet string, of a hash computation on the entire encoded component including its tag, length and value octets. Instances of OtherCertificateFormat shall be encoded in DER, whilst preserving the encoding of any signed field included in otherCert item.

NOTE 3: Use of the ats-hash-index attribute makes it possible to add additional certificate / revocation information / unsigned attribute within SignedData.certificates / SignedData.crls / unsigned attributes of the CAdES signature (for instance counter signatures), after an archive time-stamp has been applied to a signature. Its use also allows the inclusion of components required by parallel signatures at a later time.

6.5.2 The archive-time-stamp-v3 attribute

The archive-time-stamp-v3 attribute is an optional unsigned attributed defined in the present document. Several instances of this attribute may occur within an electronic signature both over time and from different TSUs. The archive-time-stamp-v3 attribute shall have a single attribute value.

The archive-time-stamp-v3 attribute is a time-stamp token of the signed document and the signature, including signed attributes, and all other essential components of the signature as protected by the ats-hash-index attribute.

The following object identifier identifies the archive-time-stamp-v3 attribute:

```
  id-aa-ets-archiveTimestampV3 OBJECT IDENTIFIER ::= { itu-t(0) identified-
organization(4) etsi(0) electronic-signature-standard(1733) attributes(2) 4 }
```

Attribute values of archive-time-stamp-v3 attribute have the ASN.1 syntax ArchiveTimeStampToken

```
  ArchiveTimeStampToken ::= TimeStampToken
```

The input for the `archive-time-stamp-v3`'s message imprint computation shall be the concatenation (in the order shown by the list below) of the signed data hash (see bullet 2 below) and certain fields in their binary encoded form without any modification and including the tag, length and value octets:

1) The `SignedData.encapContentInfo.eContentType`.

2) The octets representing the hash of the signed data. The hash is computed on the same content that was used for computing the hash value that is encapsulated within the `message-digest` signed attribute of the CAdES signature being archive-time-stamped. The hash algorithm applied shall be the same as the hash algorithm used for computing the archive time-stamp's message imprint. The inclusion of the hash algorithm in the `SignedData.digestAlgorithms` set is recommended.

3) Fields `version, sid, digestAlgorithm, signedAttrs, signatureAlgorithm`, and `signature` within the `SignedData.signerInfos`'s item corresponding to the signature being archive time-stamped, in their order of appearance.

4) A single instance of `ATSHashIndex` type (created as specified in clause 6.5.1).

The `archive-time-stamp-v3` shall include as an unsigned attribute a single `ats-hash-index`, with a single `ATSHashIndex AttributeValue` component, added to time-stamp signature.

NOTE 1: The inclusion of the `ats-hash-index` unsigned attribute's component in the process that builds the input to the computation of the archive time-stamp's message imprint ensures that all the essential components of the signature (including certificates, revocation information, and unsigned attributes) are protected by the time-stamp.

The items included in the hashing procedure and the concatenation order are shown in figure 5.

When validated, an `archive-time-stamp-v3` unsigned attribute is a proof of existence at the time indicated in its time-stamp token, of the items that have contributed to the computation of its message imprint.

NOTE 2: This proof of existence is used in validation procedures to ensure that signature validation is based on objects that truly existed in the past. This, for example, protects against a private signing key being compromised after the associated public key certificate expires resulting in the signature being considered invalid.

NOTE 3: Counter signatures held as countersignature attributes do not require independent archive time-stamps as they are protected by the archive time-stamp as an unsigned attribute.

For further information and definition of `TimeStampToken` see clause 7.4.

Before incorporating a new `archive-time-stamp-v3` attribute, the `SignedData` shall be extended to include any validation data, not already present, which is required for validating the signature being archive time-stamped. Validation data may include certificates, CRLs, OCSP responses, as required to validate any signed object within the signature including the existing signature, counter-signatures, time-stamps, OCSP response and certificates. In the case that the validation data contains a Delta CRL, then the whole the set of CRLs shall be included to provide a complete revocation list.

The present document specifies two strategies for the inclusion of validation data, depending on whether an ATSv2, other earlier form of archive time-stamp, as defined in previous versions of TS 101 733, has already been added to the `SignedData`:

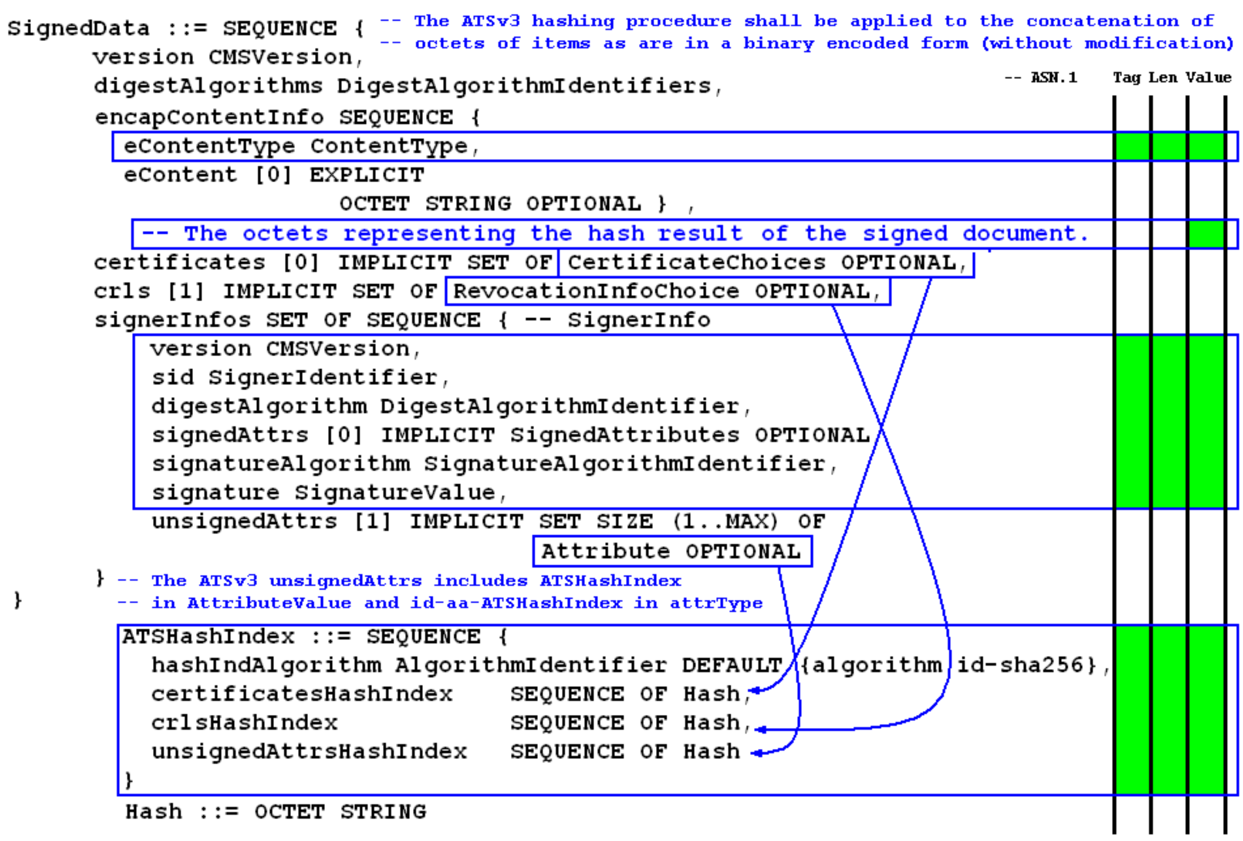
- If none of ATSv2, an earlier form of archive time-stamp or a `long-term-validation` attribute are already present, then the new validation material shall be included within the root `SignedData.certificates`, or `SignedData.crls` as applicable.

- If an ATSv2, or other earlier form of archive time-stamp, is present then the new validation material shall be included as it is specified in clause A.2.2 of the present document. Root `SignedData.certificates` and `SignedData.crls` contents shall not be modified.

NOTE 4: As specified in clause A.2.3, once long term validation data is applied to a CAdES signature an ATSv3, or any form of archive time-stamp, cannot be used. The use of `long-term-validation` attributes in signatures to which the long-term form has not already been applied is therefore deprecated.

When generating an attribute, whether initially creating the signature or extending the signature, it shall be encoded in DER, whilst preserving the encoding of any signed field included in the attribute. Extenders shall preserve the binary encoding of already present unsigned attributes and any component contributing to the archive time-stamp's message imprint computation input.

Figure 5 illustrates the hashing process:



```
SignedData ::= SEQUENCE { -- The ATSv3 hashing procedure shall be applied to the concatenation of
                          -- octets of items as are in a binary encoded form (without modification)
      version CMSVersion,
      digestAlgorithms DigestAlgorithmIdentifiers,
      encapContentInfo SEQUENCE {
        eContentType ContentType,
        eContent [0] EXPLICIT
                       OCTET STRING OPTIONAL } ,
          -- The octets representing the hash result of the signed document.
      certificates [0] IMPLICIT SET OF CertificateChoices OPTIONAL,
      crls [1] IMPLICIT SET OF RevocationInfoChoice OPTIONAL,
      signerInfos SET OF SEQUENCE { -- SignerInfo
          version CMSVersion,
          sid SignerIdentifier,
          digestAlgorithm DigestAlgorithmIdentifier,
          signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
          signatureAlgorithm SignatureAlgorithmIdentifier,
          signature SignatureValue,
          unsignedAttrs [1] IMPLICIT SET SIZE (1..MAX) OF
                                     Attribute OPTIONAL
      } -- The ATSv3 unsignedAttrs includes ATSHashIndex
}       -- in AttributeValue and id-aa-ATSHashIndex in attrType
        ATSHashIndex ::= SEQUENCE {
          hashIndAlgorithm AlgorithmIdentifier DEFAULT {algorithm id-sha256},
          certificatesHashIndex      SEQUENCE OF Hash,
          crlsHashIndex              SEQUENCE OF Hash,
          unsignedAttrsHashIndex     SEQUENCE OF Hash
        }
        Hash ::= OCTET STRING
```

Figure 5: Hashing process

6.5.2.1 Validation of the `archive-time-stamp-v3` attribute

When validating the `archive-time-stamp-v3`, first the contained `ats-hash-index` shall be validated. All the hash values of all of the certificates, revocation information and unsigned attributes are recalculated. Only those which match one of the hash values in `ATSHashIndex` are known to be protected by the corresponding archive time-stamp. The `ats-hash-index` is invalid if it contains a reference for which the original value is not found, i.e.

- a reference represented by an entry in `certificatesHashIndex` which corresponds to no instance of `CertificateChoices` within `certificates` field of the root `SignedData`,
- a reference represented by an entry in `crlsHashIndex` which corresponds to no instance of `RevocationInfoChoice` within `crls` field of the root `SignedData`, or
- a reference represented by an entry in `unsignedAttrsHashIndex` which corresponds to no instance of `Attribute` within `unsignedAttrs` field of the `SignerInfo`.

Once the `ats-hash-index` is validated, the `archive-time-stamp-v3` can be validated by recalculating the message imprint in the same way as in the creation of the attribute.

7 Conformance requirements

The current document defines several conformance levels. In this clause we define the requirements of the four main levels:

- CAdES-Basic Electronic Signature (CAdES-BES)
- CAdES-Explicit Policy-based Electronic Signature (CAdES-EPES)
- Electronic Signature with time (CAdES-T)
- CAdES-A with archive-time-stamp (CAdES-A)

NOTE: These conformance levels are more general than the levels defined in the baseline profile, see part 2 of the current series [i.6].

The normative annex C describes additional conformance levels.

An implementation claiming to be conformant to a specific level of the current document shall fulfil the corresponding requirements defined in the current clause or in annex C.

7.1 CAdES-Basic Electronic Signature (CAdES-BES)

A signature corresponding to CAdES-BES level, shall, at a minimum, consisting of the following components:

- The general CMS syntax and content type, as defined in RFC 5652 [7] (see clauses 5 and 5.1).
- CMS `SignedData`, as defined in RFC 5652 [7], with the version set to either 1 or 3 as specified in clause 5.1 of RFC 5652 [7] (see note 2) and at least one `SignerInfo` present (see clauses 5.2 to 5.4)
- The following CMS attributes, as defined in RFC 5652 [7]:
    - `content-type`; this shall always be present (see clause 6.1.1); and
    - `message-digest`; this shall always be present (see clause 6.1.2).
- One of the following attributes, as defined in the present document:
    - `signing-certificate`: as defined in clause 6.2.2.1; or
    - `signing-certificate` v2 as defined in clause 6.2.2.2.

NOTE 1: Earlier versions of TS 101 733 used the other signing-certificate attribute (see clause A.2.1). Its use is now deprecated, since the structure of the signing-certificate v2 attribute is simpler than the other signing-certificate attribute.

NOTE 2: Clause 5.1 of RFC 5652 [7] requires that, the CMS `SignedData` version be set to 3 if certificates from `SignedData` are present AND (any version 1 attribute certificates are present OR any `SignerInfo` structures are version 3 OR `eContentType` from `encapContentInfo` is other than `id-data`). Otherwise, the CMS `SignedData` version is required to be set to 1.

This clause defines the minimal set of attributes mandated to be conformant to the current document. A signature policy may mandate that other signed attributes be present.

7.2 CAdES-Explicit Policy-based Electronic Signature (CAdES-EPES)

A system supporting Policy-based signers, and thus corresponding to CAdES-EPES level according to the present document, shall, at a minimum consists of the following components:

- All components needed for CAdES-BES level

- The following attribute, as defined in clause 6.2.9.1:
  - `signature-policy-identifier`; this shall always be present.

7.3 Electronic Signature with time (CAdES-T)

A signature corresponding to CAdES-T level shall be build upon CAdES-BES or CAdES-EPES. In addition, there shall exists a trusted time associated with the signature. The trusted time may be provided by:

- a `time-stamp attribute` as an unsigned attribute (see clause 6.3); and
- a time-mark of the electronic signature provided by a Trust Service Provider.

7.4 CAdES-A with archive-time-stamp (CAdES-A)

A signature corresponding to CAdES-A level shall be build upon a signature having at least level CAdES-BES or CAdES-EPES.

The archive time-stamp gives an assurance when the signature existed already. However, it is highly recommended to build CAdES-A upon CAdES-T, especially if the archiving is done long after the creation of the signature.

In addition it shall contain:

- one or more `archive-time-stamp-v3` attributes (see clause 6.5.2) to protect the complete certificate and revocation data, or/and
- one or more archive time-stamp or long-term validation time-stamps defined in previous version of TS 101 733 [i.1], see also clauses A.2.2 and A.2.3.

NOTE: The second bullet is needed to allow the archival of legacy signatures.

When a `long-term-validation` attribute is not present, applications claiming conformance to the present document shall generate an `archive-time-stamp-v3` attributes (see clause 6.5.2) whenever a new archive time-stamp is required within the CAdES signature.

When a `long-term-validation` attribute is already present, applications claiming conformance to the present document shall generate a `long-term-validation` attributes (see clause A.2.3) whenever the archival time-stamp must be renewed.

Annex A (normative): Additional Attributes Specification

A.1 Attributes for validation data

A.1.1 Certificates validation data

A.1.1.1 The `complete-certificate-references` attribute

The `complete-certificate-references` attribute is an optional unsigned attribute. At most one single instance of this attribute shall be present. If the attribute is present, it shall contain exactly one component of `AttributeValue` type.

It references the full set of CA certificates that have been used to validate the electronic signature up to (but not including) the signer's certificate.

NOTE 1: The signer's certificate is referenced in the signing certificate attribute (see clause 6.2.2).

```
id-aa-ets-certificateRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 21}
```

The `complete-certificate-references` attribute value has the ASN.1 syntax `CompleteCertificateRefs`.

```
CompleteCertificateRefs ::=  SEQUENCE OF OtherCertID

OtherCertID ::= SEQUENCE {
    otherCertHash            OtherHash,
    issuerSerial             IssuerSerial OPTIONAL }

OtherHash ::= CHOICE {
    sha1Hash OtherHashValue,  -- This contains a SHA-1 hash
    otherHash OtherHashAlgAndValue}

OtherHashValue ::= OCTET STRING

OtherHashAlgAndValue ::= SEQUENCE {
    hashAlgorithm   AlgorithmIdentifier,
    hashValue       OtherHashValue }
```

The `IssuerSerial` shall be present in `OtherCertID`. The `certHash` shall match the hash of the certificate referenced.

NOTE 2: Copies of the certificate values may be held using the `certificate-values` attribute, defined in clause A.1.1.2 or within `SignedData.certificates`.

This attribute may include references to the certification chain for any TSU that provides time-stamp tokens. In this case, the unsigned attribute shall be added to the `signedData` of the relevant time-stamp token as an `unsignedAttrs` in the `signerInfos` field.

A.1.1.2 The `certificate-values` attribute

The `certificate-values` attribute is an optional unsigned attribute. At most one single instance of this attribute shall be present. If the attribute is present, it shall contain exactly one component of `AttributeValue` type.

It holds the values of certificates referenced in the `complete-certificate-references` attribute. Certificates that are referenced in `complete-certificate-references` and which are already stored in `SignedData.certificates`, need not to be included.

NOTE: If an attribute certificate is used, it is not provided in this structure but will be provided by the signer as a `signer-attributes` attribute (see clause 6.2.6).

The following object identifier identifies the `certificate-values` attribute:

```
id-aa-ets-certValues OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 23}
```

The `certificate-values` attribute value has the ASN.1 syntax `CertificateValues:`

```
CertificateValues ::=  SEQUENCE OF Certificate
```

`Certificate` is defined in the X.509 v3 certificate basis syntax in Recommendation ITU-T X.509 [14].

This attribute may be used to contain the certificate information required for the following forms of extended electronic signature: CAdES-X Long, CAdES X-Long Type 1, and CAdES-X Long Type 2, and CAdES-A containing a least one `archive-time-stamp` (ATSv2), see clauses B.3.1 and B.3.2 for an illustration of these forms of electronic signature.

This attribute may include the certification information for any TSUs that have provided the time-stamp tokens, if these certificates are not already included in the TSTs as part of the TSUs signatures. In this case, the unsigned attribute shall be added to the `signedData` of the relevant time-stamp token.

A.1.2 Revocation validation data

A.1.2.1 The `complete-revocation-references` attribute

The `complete-revocation-references` attribute is an optional unsigned attribute At most one single instance of this attribute shall be present. If the attribute is present, it shall contain exactly one component of `AttributeValue` type.

It references the full set of the CRL, or OCSP responses that have been used in the validation of the signer, and CA certificates used in Electronic Signatures with Complete validation data.

The following object identifier identifies the `complete-revocation-references` attribute:

```
id-aa-ets-revocationRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 22}
```

The `complete-revocation-references` attribute value has the ASN.1 syntax `CompleteRevocationRefs`

```
CompleteRevocationRefs ::=  SEQUENCE OF CrlOcspRef

CrlOcspRef ::= SEQUENCE {
    crlids          [0] CRLListID   OPTIONAL,
    ocspids         [1] OcspListID  OPTIONAL,
    otherRev        [2] OtherRevRefs OPTIONAL
}
```

`CompleteRevocationRefs` shall contain one `CrlOcspRef` for the `signing-certificate`, followed by one for each `OtherCertID` in the `CompleteCertificateRefs` attribute. The second and subsequent `CrlOcspRef` fields shall be in the same order as the `OtherCertID` to which they relate. At least one of `CRLListID` or `OcspListID` or OtherRevRefs should be present for all but the "trusted" CA of the certificate path.

```
CRLListID ::=  SEQUENCE {
    crls        SEQUENCE OF CrlValidatedID }

CrlValidatedID ::=  SEQUENCE {
     crlHash                    OtherHash,
     crlIdentifier              CrlIdentifier OPTIONAL }

CrlIdentifier ::= SEQUENCE {
    crlissuer                  Name,
    crlIssuedTime              UTCTime,
    crlNumber                  INTEGER OPTIONAL }

OcspListID ::=  SEQUENCE {
    ocspResponses        SEQUENCE OF OcspResponsesID }

OcspResponsesID ::=  SEQUENCE {
    ocspIdentifier              OcspIdentifier,
    ocspRefHash                 OtherHash    OPTIONAL
}

OcspIdentifier ::= SEQUENCE {
    ocspResponderID     ResponderID,    -- As in OCSP response data
    producedAt          GeneralizedTime -- As in OCSP response data
}

OtherRevRefs ::= SEQUENCE {
    otherRevRefType OtherRevRefType,
    otherRevRefs    ANY DEFINED BY otherRevRefType
 }

OtherRevRefType ::= OBJECT IDENTIFIER
```

When creating a `crlValidatedID`, the `crlHash` is computed over the entire DER encoded CRL including the signature. The `crlIdentifier` should normally be present unless the CRL can be inferred from other information.

The `crlIdentifier` is to identify the CRL using the issuer name and the CRL issued time, which shall correspond to the time `thisUpdate` contained in the issued CRL, and if present, the `crlNumber`. In the case that the identified CRL is a Delta CRL, then references to the set of CRLs to provide a complete revocation list shall be included.

The `OcspIdentifier` is to identify the OCSP response using the issuer name and the time of issue of the OCSP response, which shall correspond to the time produced as contained in the issued OCSP response. In previous versions of the standard, the `ocspRefHash` field was optional. In order to provide backward compatibility, the ASN.1 structure is not changed, however, it is strongly recommended that implementations include this element. Implementations verifying a signature may choose to accept signatures without this element, but should be warned that its absence makes OCSP responses substitutions attacks possible, if for instance OCSP responder keys are compromised. Implementations choosing to accept signatures without this element may use out-of-band mechanisms to ensure that none of the OCSP responder keys have been compromised at the time of validation.

NOTE 1: Copies of the CRL and OCSP responses values may be held using the `revocation-values` attribute defined in clause A.1.2.2 or within `SignedData.crls`.

NOTE 2: It is recommended that this attribute be used in preference to the `OtherRevocationInfoFormat` specified in RFC 5652 [7] to maintain backwards compatibility with the earlier version of TS 101 733.

The syntax and semantics of other revocation references are outside the scope of the present document. The definition of the syntax of the other form of revocation information is as identified by `OtherRevRefType`.

This attribute indicates that the verifier has taken due diligence to gather the available revocation information. The references stored in this attribute can be used to retrieve the referenced information, if not stored in the CMS structure, but somewhere else.

This attribute may include the references to the full set of the CRL, ACRL, or OCSP responses that have been used to verify the certification chain for any TSUs that provide time-stamp tokens. In this case, the unsigned attribute shall be added to the `signedData` of the relevant time-stamp token.

A.1.2.2 The `revocation-values` attribute

The `revocation-values` attribute is an optional unsigned attribute. At most one single instance of this attribute shall be present. If the attribute is present, it shall contain exactly one component of `AttributeValue` type.

It holds the values of CRLs and OCSP referenced in the `complete-revocation-references` attribute. CRLs and OCSP responses that are referenced in `complete-revocation-references` and which are already stored in `SignedData.crls`, need not to be included.

NOTE: It is recommended that this attribute be used in preference to the `OtherRevocationInfoFormat` specified in RFC 5652 [7] to maintain backwards compatibility with the earlier version of the present document.

The following object identifier identifies the `revocation-values` attribute:

```
id-aa-ets-revocationValues OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 24}
```

The `revocation-values` attribute value has the ASN.1 syntax `RevocationValues`

```
RevocationValues ::=  SEQUENCE {
   crlVals          [0] SEQUENCE OF CertificateList OPTIONAL,
   ocspVals         [1] SEQUENCE OF BasicOCSPResponse OPTIONAL,
   otherRevVals     [2] OtherRevVals OPTIONAL}

OtherRevVals ::= SEQUENCE {
    otherRevValType OtherRevValType,
    otherRevVals    ANY DEFINED BY OtherRevValType
 }

OtherRevValType ::= OBJECT IDENTIFIER
```

The syntax and semantics of the contents of `OtherRevVals` field are outside the scope of the present document. The definition of the syntax of the other form of revocation information is as identified by `OtherRevRefType`.

`CertificateList` is defined in the X.509 v2 CRL syntax in Recommendation ITU-T X.509 [14].

`BasicOCSPResponse` is defined in RFC 6960 [9].

This attribute is used to contain the revocation information required for the following forms of extended electronic signature: CAdES-X Long, CAdES X-Long Type 1, and CAdES-X Long Type 2, see clauses B.3.1 and B.3.2 for an illustration of these forms of electronic signature

This attribute may include the values of revocation data including CRLs and OCSPs for any TSUs that have provided the time-stamp tokens, if these certificates are not already included in the TSTs as part of the TSUs signatures. In this case, the unsigned attribute shall be added to the `signedData` of the relevant time-stamp token.

A.1.3 The `attribute-certificate-references` attribute

The `attribute-certificate-references` attribute is an optional unsigned attribute. At most one single instance of this attribute shall be present. If the attribute is present, it shall contain exactly one component of `AttributeValue` type.

This attribute is only used when a user attribute certificate is present in the electronic signature.

It references the full set of AA certificates that have been used to validate the attribute certificate when present in the signature. The attribute may also contain references to the CA certificates within the certification paths of the attribute certificates used in the validation, if not incorporated elsewhere within the CAdES signature.

EDITOR NOTE: previous versions of CAdES did not make this issue clear. Feedback is requested from stakeholders regarding the suitability of this amendment.

If `signedAssertsions` (within `signer-attributes-v2`, clause 6.2.6.2) are used, references to the certificates used to validate the signature of the assertion and which are not incorporated elsewhere within the CAdES signature, may be stored here as well.

EDITOR NOTE: it may also contain material for signed assertions. Feedback kindly requested from stakeholders.

```
id-aa-ets-attrCertificateRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 44}
```

The `attribute-certificate-references` attribute value has the ASN.1 syntax `AttributeCertificateRefs`:

```
AttributeCertificateRefs ::=  SEQUENCE OF OtherCertID
```

`OtherCertID` is defined in clause A.2.1.

NOTE: Copies of the certificate values referenced here may be held using the `certificate-values` attribute defined in clause A.1.1.2 or within `SignedData.certificates`. The attribute certificate itself is stored in the `signer-attributes` as defined in clause 6.2.6.

A.1.4 The `attribute-revocation-references` attribute

The `attribute-revocation-references` attribute is an optional unsigned attribute. At most one single instance of this attribute shall be present. If the attribute is present, it shall contain exactly one `AttributeValue`.

This attribute is only used when a user attribute certificate is present in the electronic signature and when that attribute certificate can be revoked.

It contains references to ACRL, CRL or OCSP responses that have been used in the validation of the attribute certificate(s) present in the signature, if they are not incorporated elsewhere within the CAdES signature. It may also contain references to revocation data that have been used to validate the signatures of signed assertions within `signer-attributes-v2`, clause 6.2.6.2, if they are not incorporated elsewhere within the CAdES signature. This attribute can be used to illustrate that the verifier has taken due diligence of the available revocation information. In the case that the identified ACRL or CRL is a Delta CRL, then references to the set of CRLs to provide a complete revocation list shall be included.

The following object identifier identifies the `attribute-revocation-references` attribute:

```
id-aa-ets-attrRevocationRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 45}
```

The `attribute-revocation-references` attribute value has the ASN.1 syntax `AttributeRevocationRefs:`

```
AttributeRevocationRefs ::=  SEQUENCE OF CrlOcspRef
```

A.1.5 Time-stamps on references to validation data

A.1.5.1 The `time-stamped-certs-crls-references` attribute

The `time-stamped-certs-crls-references` attribute is an optional unsigned attribute. Several instances of this attribute may occur with an electronic signature from different TSAs.

The following object identifier identifies the `time-stamped-certs-crls-references` attribute:

```
id-aa-ets-certCRLTimestamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 26}
```

The attribute value has the ASN.1 syntax `TimestampedCertsCRLs`:

```
TimestampedCertsCRLs ::= TimeStampToken
```

This attribute encapsulates a time-stamp token, whose `messageImprint` field shall be the hash of the concatenated values (without the type or length encoding for that value) of the following data objects, as present within the electronic signature:

- `complete-certificate-references` attribute; and
- `complete-revocation-references` attribute.

NOTE 1: It is recommended that the attributes being time-stamped be encoded in DER. If DER is not employed, then the binary encoding of the ASN.1 structures being time-stamped should be preserved to ensure that the recalculation of the data hash is consistent.

NOTE 2: Each attribute is included in the hash with the `attrType` and `attrValues` (including type and length) but without the type and length of the outer SEQUENCE.

This attribute is used to protect against CA key compromise. It is used in the following forms of extended electronic signature: CAdES-X Type 2 and CAdES-X Long Type 2; see clauses B.2.2 and B.3.2 for an illustration of these forms of electronic signature."

A.1.5.2 The `CAdES-C-timestamp` attribute

The `CAdES-C-time-stamp` attribute is an optional unsigned attribute. Several instances of this attribute may occur with an electronic signature from different TSAs.

The following object identifier identifies the `CAdES-C-Timestamp` attribute:

```
id-aa-ets-escTimeStamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 25}
```

The `CAdES-C-timestamp` attribute value has the ASN.1 syntax `ESCTimeStampToken`:

```
ESCTimeStampToken ::= TimeStampToken
```

This attribute encapsulates a time-stamp token, whose `messageImprint` field shall be the hash of the concatenated values (without the ASN.1 type or length encoding for that value) of the following data objects:

- OCTETSTRING of the `signature` field within `SignerInfo`;
- `signature-time-stamp`
- `complete-certificate-references` attribute; and
- `complete-revocation-references` attribute

NOTE 1: It is recommended that the attributes being time-stamped be encoded in DER. If DER is not employed, then the binary encoding of the ASN.1structures being time-stamped should be preserved to ensure that the recalculation of the data hash is consistent.

NOTE 2: Each attribute is included in the hash with the attrType and attrValues (including type and length) but without the type and length of the outer SEQUENCE.

This attribute is used to protect against CA key compromise. It is used for the time-stamping of the complete electronic signature (CAdES-C). It is used in the following forms of extended electronic signature; CAdES-X Type 1 and CAdES-X Long Type 1; see clause B.2.1 and B.3.2 for an illustration of these forms of electronic signature.

A.2 Obsolete attributes

A.2.1 The `other-signing-certificate` attribute

Earlier versions of TS 101 733 used the other signing-certificate attribute as an alternative to the ESS signing-certificate when hashing algorithms other than SHA-1 were being used. Its use is now deprecated, since the structure of the signing-certificate-v2 attribute is simpler.

Its description is however still present in the present document for backwards compatibility. There shall be never more than one instance of this attribute.

```
id-aa-ets-otherSigCert OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-aa(2) 19 }
```

The `other-signing-certificate` attribute value has the ASN.1 syntax `OtherSigningCertificate`:

```
OtherSigningCertificate ::=  SEQUENCE {
    certs        SEQUENCE OF OtherCertID,
    policies     SEQUENCE OF PolicyInformation OPTIONAL
```

```
                -- NOT USED IN THE PRESENT DOCUMENT }
```

OtherCertID is defined in clause A.1.1.1.

A.2.2 The archive-time-stamp attribute

The archive-time-stamp (ATSv2) attribute is deprecated. ATSv2 is included for backward compatibility. Systems may extend the lifetime of signatures incorporating ATSv2 by incorporating ATSv3 as described in clause 6.5.2.

The archive-time-stamp attribute is an unsigned attribute. Several instances of this attribute may occur with an electronic signature both over time and from different TSUs.

The archive-time-stamp (ATSv2) attribute encapsulates a time-stamp token of many of the elements of the signedData in the electronic signature.

If the certificate-values and revocation-values attributes are not present in the CAdES-BES or CAdES-EPES, then they shall be added to the electronic signature prior to computing the archive time-stamp token. The

The following object identifier identifies the nested archive-time-stamp attribute:

```
id-aa-ets-archiveTimestampV2 OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2)  48}
```

Archive-time-stamp attribute values have the ASN.1 syntax ArchiveTimeStampToken

```
ArchiveTimeStampToken ::= TimeStampToken
```

The value of the messageImprint field within TimeStampToken shall be a hash of the concatenation of:

- the encapContentInfo element of the SignedData sequence including the ASN.1 tag and length;
- any external content being protected by the signature, if the eContent element of the encapContentInfo is omitted;
- the Certificates and crls elements of the SignedData sequence, when present, including the ASN.1 tag and length; and
- **all** data elements in the SignerInfo sequence including all signed and unsigned attributes, each time including the ASN.1 tag and length.

NOTE 1: An alternative archiveTimestamp attribute, identified by an object identifier { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 27, is defined in prior versions of TS 101 733. The archiveTimestamp attribute, defined in versions of TS 101 733 prior to 1.5.1, is **not** compatible with the attribute defined in the present document. The archiveTimestamp attribute, defined in versions 1.5.1 to 1.6.3 of TS 101 733, is compatible with the present document if the content is internal to encapContentInfo. Unless the version of TS 101 733 employed by the signing party is known by all recipients, use of the archiveTimestamp attribute defined in prior versions of TS 101 733 is deprecated.

NOTE 2: Counter signatures held as countersignature attributes do not require independent archive time-stamps as they are protected by the archive time-stamp against the containing SignedData structure.

NOTE 3: Unless DER is used throughout, it is recommended that the binary encoding of the ASN.1 structures being time-stamped be preserved when being archived to ensure that the recalculation of the data hash is consistent.

NOTE 4: The hash is calculated over the concatenated data elements as received /stored including the Type and Length encoding.

NOTE 5: Whilst it is recommended that unsigned attributes be DER encoded, it cannot generally be so guaranteed except by prior arrangement.

The format of a `TimeStampToken` type is defined in RFC 3161 [3] and profiled in TS 101 861 [i.2].

The timestamp should be created using stronger algorithms (or longer key lengths) than in the original electronic signatures and weak algorithm (key length) timestamps.

The `ArchiveTimeStamp` will be added as an unsigned attribute in the `SignerInfo` sequence. For the validation of one `ArchiveTimeStamp`, the data elements of the `SignerInfo` shall be concatenated, excluding all later `ArchivTimeStampToken` attributes.

Certificates and revocation information required to validate the ArchiveTimeStamp shall be provided by one of the following methods:

- the TSU provides the information in the `SignedData` of the timestamp token;
- adding the `complete-certificate-references` attribute and the `complete-revocation-references` attribute of the TSP as an unsigned attribute within TimeStampToken, when the required information is stored elsewhere; or
- adding the `certificate-values` attribute and the `revocation-values` attribute of the TSP as an unsigned attribute within TimeStampToken, when the required information is stored elsewhere.

A.2.3 The `long-term-validation` attribute

The use of the `long-term-validation` attribute is deprecated unless a `long-term-validation` attributes is already present in a signature.

The `long-term-validation` attribute is an optional unsigned attribute. Several instances of this attribute may occur within the list of unsigned attributes, each instance being used in the computation of subsequently added instances.

The `long-term-validation` attribute is a proof of existence (PoE) at a past date, computed over many of the elements of the `SignedData` in the electronic signature.

Once a `long-term-validation` attribute has been added to an electronic signature, no other attribute may be added except other `long-term-validation` attributes, and no attribute value may be altered except possibly the last added `long-term-validation` attribute.

The following object identifier identifies the nested `long-term-validation` attribute:

```
id-aa-ets-longTermValidation OBJECT IDENTIFIER ::= { itu-t(0) identified-organization(4) etsi(0)
       electronic-signature-standard (1733) attributes(2) 2 }
```

Each instance of this attribute shall be encoded in its own `Attribute` structure, the `attrValues` field then consisting of a `SET` of exactly one value conforming to this syntax:

```
LongTermValidation ::= SEQUENCE {
        poeDate    GeneralizedTime,
        poeValue   CHOICE {
                timeStamp         [0] EXPLICIT TimeStampToken,
                evidenceRecord    [1] EXPLICIT EvidenceRecord
        } OPTIONAL,
        extraCertificates    [0] IMPLICIT CertificateSet OPTIONAL,
        extraRevocation      [1] IMPLICIT RevocationInfoChoices OPTIONAL
}
```

The `poeDate` records the approximate date at which this attribute is added to an electronic signature; it shall be ulterior to all `poeDate` values for previously added `long-term-validation` attributes.

The PoE itself proves the existence of some data at a precise past date, which is recorded within the PoE structure; accessing that PoE date requires decoding the PoE, a process which depends on the PoE type. The `poeDate` field should contain a copy of the PoE creation date; however, it is acceptable that the `poeDate` value is not strictly equal to the PoE creation date. This is expected in case the system which assembles the attribute instance obtains the PoE from an external system, and cannot decode it itself. In that situation, the system which adds the attribute instance should use the current date, which may be off from the PoE actual creation date by a small amount.

For long-term signature verification purposes, the `poeDate` field should be used only to infer the order in which the `long-term-validation` attribute instances were added; only the PoE date specified within the PoE itself is actually "proven". Verifiers should not fail in case the `poeDate` field is not equal to the PoE creation date.

The `extraCertificates` and `extraRevocation` fields are meant to receive extra validation objects which may be used to help validate the document signature, various time-stamps included in the signature or a PoE on an older `long-term-validation` attribute instance.

The PoE value shall be either a `TimeStampToken` as defined in clause 5.6.1 or an `EvidenceRecord` as defined in RFC 4998 [4]. The format of `TimeStampToken` contained within these `EvidenceRecord` is as defined in clause 5.6.1.

A time-stamp includes (in the `messageImprint` field of the `TimeStampToken` structure) a hash value computed over many of the `SignedData` elements. In the case of the `long-term-validation` attribute, a tree-like structure is used to process some elements which are, at the ASN.1 level, encoded with a `SET`, hence orderless.

The computation of the aforementioned hash value is done through "tree-hashing", which is a way to hash a set of data elements with a hash function h, such that the *ordering* of these elements is irrelevant and does not impact the final value. Tree-hashing will be used to process elements of the `SignedData` which are in no particular order (they are part of a `SET OF`) and might be subject to spurious reordering.

Given a hash function h, and a set of individual data elements D_i (without particular order), the elements shall be tree-hashed as follows:

- Each element D_i is hashed by itself with h, yielding a value V_i.
- The V_i are concatenated in ascending lexicographic order:
  - In lexicographic order, a byte sequence a_j is said to be lower than a byte sequence b_j if there exists an index k such that $a_j = b_j$ for all $j < k$, and $a_k < b_k$. Byte values are integers between 0 and 255, inclusive.
  - If there are r data elements, and the hash function outputs n bytes, then the concatenation results in a sequence of rn bytes.
  - Duplicates, if any, are not collapsed: in that case, we keep all the identical hash values in the concatenation result.
- The concatenation result is then itself hashed with h. That final hash output is the tree-hash result.

NOTE 1: If the tree-hash is applied on an empty sequence of data elements, the resulting output is $h(z)$ where z is a bit string of length zero.

To obtain the hash value used as input to the PoE creation (e.g. the `messageImprint` value for a time-stamp), the following process shall be applied:

- A hash function h is used; it shall be one of the hash functions identified in the `digestAlgorithms` field of the `SignedData`.
- A hash value H_e is computed over the DER encoding of the `eContentType` field of the `encapContentInfo` of the `SignedData`. The Type and Length are included in the input to the hash function for the computation of H_e.
- A hash value H_m is computed over the value of the `eContent` field of the `encapContentInfo` of the `SignedData`. Only the actual data bytes are hashed, not the Type, Length, or any substructure if the `OCTET STRING` is constructed. If the `eContent` field is absent, the hash value is computed on the external protected content. H_m is the value which goes into the `message-digest` signed attribute, if the same hash function h is used.
- A tree-hash value H_c is computed over the elements of the `certificates` field of the `SignedData` and the `extraCertificates`. The "data elements" for the tree-hash are the encodings (including Type and

Length) of each individual `CertificateChoices` values, from both the `certificates` field, and the set of extra certificates which will be included in the new `long-term-validation` instance. The encodings as received / stored should be used.

NOTE 2: The Type and Length of the outer `SET`s are not hashed in any way. If both the `certificates` (from `SignedData`) and `extraCertificates` fields (from the to-be-added `long-term-validation` attribute) are omitted, H_c is nonetheless computed as the tree-hash of an empty set of elements (i.e. $H_c = h(z)$, the hash of the bit string of length zero).

- A tree-hash value H_r is computed over the elements of the `crls` and `extraRevocation` fields (from `SignedData` and to-be-added `long-term-validation` attribute instance, respectively) with the same method than the one used for H_c (data elements are the individual `RevocationInfoChoice` values).

- The DER-encoded `version`, `sid` and `digestAlgorithm` fields are concatenated and then hashed together, in that order, yielding H_v. The Type and Length encodings of all three fields are included (but not any Type or Length encoding from the encapsulating `SEQUENCE`).

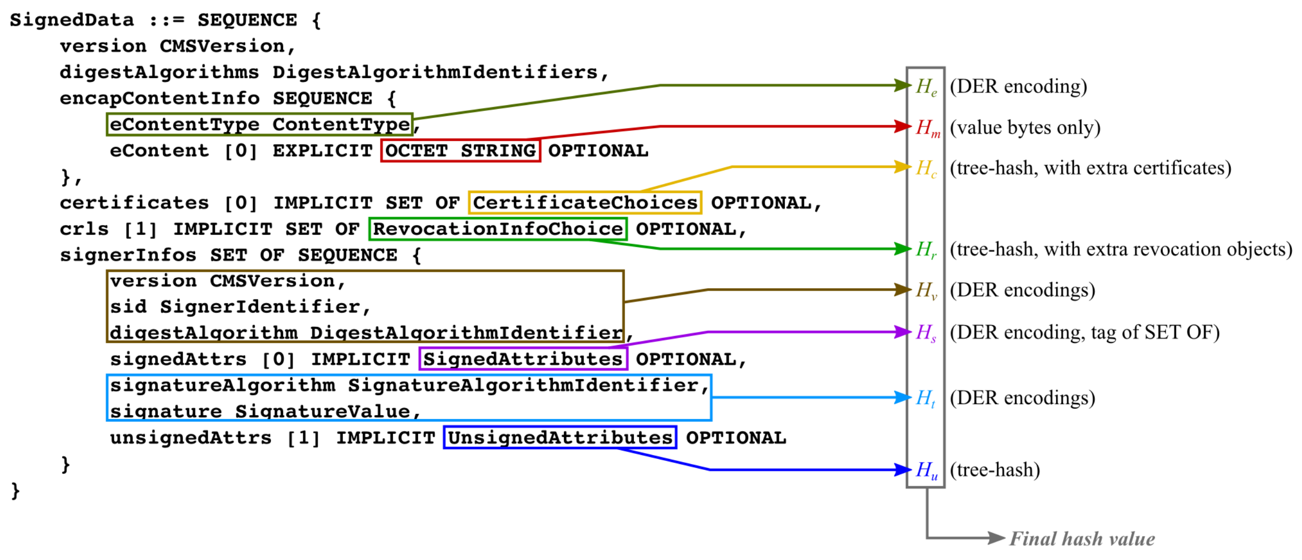
- A hash value H_s is computed over the encoding of the signed attributes, in the same way as defined for CMS signatures (RFC 5652 [7], clause 5.4): *h* is applied over the DER encoding of the `signedAttrs` field, except that a `SET OF` tag is used instead of the `IMPLICIT [0]` which is applied on that field as part of the `SignedData` structure. H_s is the input of the signature generation function which was used to sign the document in the first place, if the same hash function *h* is used.

- The DER-encoded `signatureAlgorithm` and `signature` fields are concatenated and then hashed together, in that order, yielding H_t. The Type and Length encodings of both fields are included.

- A tree-hash value H_u is computed over the unsigned attributes. Data elements are the encoded `Attribute` structures, as they are received / stored. The Type and Length of each individual `Attribute` structure are included.

- The hash values H_e, H_m, H_c, H_r, H_v, H_s, H_t and H_u are concatenated in that order. The resulting string is the input to the time-stamp generation process; in other words, the `messageImprint` field contains the value $h(H_e||H_m||H_c||H_r||H_v||H_s||H_t||H_u)$ (where "||" denotes concatenation).

Figure 12 illustrates the hashing process:



```
SignedData ::= SEQUENCE {
    version CMSVersion,
    digestAlgorithms DigestAlgorithmIdentifiers,
    encapContentInfo SEQUENCE {
        eContentType ContentType,
        eContent [0] EXPLICIT OCTET STRING OPTIONAL
    },
    certificates [0] IMPLICIT SET OF CertificateChoices OPTIONAL,
    crls [1] IMPLICIT SET OF RevocationInfoChoice OPTIONAL,
    signerInfos SET OF SEQUENCE {
        version CMSVersion,
        sid SignerIdentifier,
        digestAlgorithm DigestAlgorithmIdentifier,
        signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
        signatureAlgorithm SignatureAlgorithmIdentifier,
        signature SignatureValue,
        unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL
    }
}
```

H_e (DER encoding)
H_m (value bytes only)
H_c (tree-hash, with extra certificates)
H_r (tree-hash, with extra revocation objects)
H_v (DER encodings)
H_s (DER encoding, tag of SET OF)
H_t (DER encodings)
H_u (tree-hash)
Final hash value

Figure A.1: Hashing process

If the PoE is an evidence record, then the process described above shall be done for every algorithm h_i in `EvidenceRecord.digestAlgorithms`, which will lead to a final digest computed by h_i. These final values for the different algorithms will be used as leaves in the hash tree building process of the `ArchiveTimeStamp` of the evidence record as described in RFC 4998 [4]. If `EvidenceRecord.digestAlgorithms` contains only a single algorithm, the hash tree in the ArchiveTimeStamp will contain only a single leave.

All the hash functions contained in `EvidenceRecord.digestAlgorithms` shall also be included in the `digestAlgorithms` field of the `SignedData`.

Annex B (normative): CAdES signature forms with references and obsolete attributes

B.1 Electronic signatures with complete validation data references (CAdES-C)

Electronic Signature with Complete validation data references (CAdES-C), in accordance with the present document, adds to the CAdES-T the `complete-certificate-references` and `complete-revocation-references` attributes, as defined by the present document. Clauses A.1.1.1 and A.1.2.1 provide the specification details. Storing the references allows the values of the certification path and the CRLs or OCSP responses to be stored elsewhere, reducing the size of a stored electronic signature format.

This form represents an AdES-C signature as described in ETSI EN 319 102 [i.5].

Electronic signatures, with the additional validation data forming the CAdES-C, are illustrated in figure B.1.

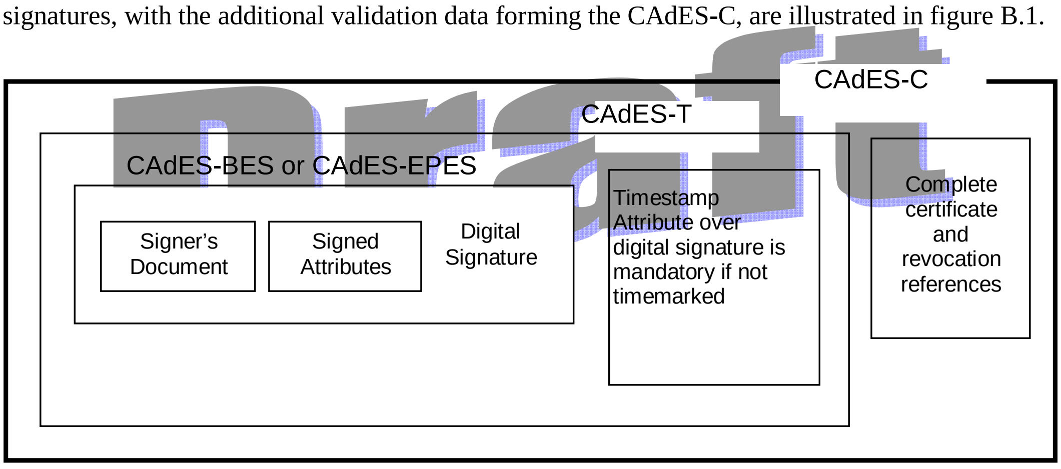

CAdES-C
CAdES-T
CAdES-BES or CAdES-EPES
Signer's Document
Signed Attributes
Digital Signature
Timestamp Attribute over digital signature is mandatory if not timemarked
Complete certificate and revocation references

Figure B.1: Illustration of CAdES-C format

NOTE 1: The complete certificate and revocation references are added to the CAdES-T as an unsigned attribute.

NOTE 2: As a minimum, the signer will provide the CAdES-BES or, when indicating that the signature conforms to an explicit signing policy, the CAdES-EPES.

NOTE 3: The signer and TSP could provide the CAdES-C to minimize this risk, and when the signer does not provide the CAdES-C, the verifier should create the CAdES-C when the required component of revocation and validation data become available; this may require a grace period (see EN 319 102 [i.5]).

NOTE 4: Time-stamp tokens may themselves include unsigned attributes required to validate the time-stamp token.

If attributes certificates are used, then the following attributes may be present:

- `attribute-certificate-references` attribute, defined in clause A.1.3;
- `attribute-revocation-references` attribute, as defined in clause A.1.4.

B.2 Extended signatures with time forms (CAdES-X)

EXtended signatures with time forms (CAdES-X) are build upon CAdES-C by adding `CAdES-C-time-stamped-certs-crls-references` or `CAdES-C-Timestamp` attributes. There exits two different forms, CAdES-X Type 1 and CAdES-X Type 2, which different in the scope of the protective time-stamp. Both CAdES-X Type 1 and CAdES-X Type 2 counter the same threats, and the usage of one or the other depends on the environment.

B.2.1 CAdES-X Type 1

Extended format with time type 1 (**CAdES-X Type 1**), in accordance with the present document, is illustrated in figure B.2. It adds to CAdES-C signature one or more `CAdES-C-Timestamp` attributes. Clause A.1.5.2 provides the specification details on the `CAdES-C-Timestamp` attribute.

This form represents an AdES-X Type 1 signature as described in ETSI EN 319 102 [i.5].

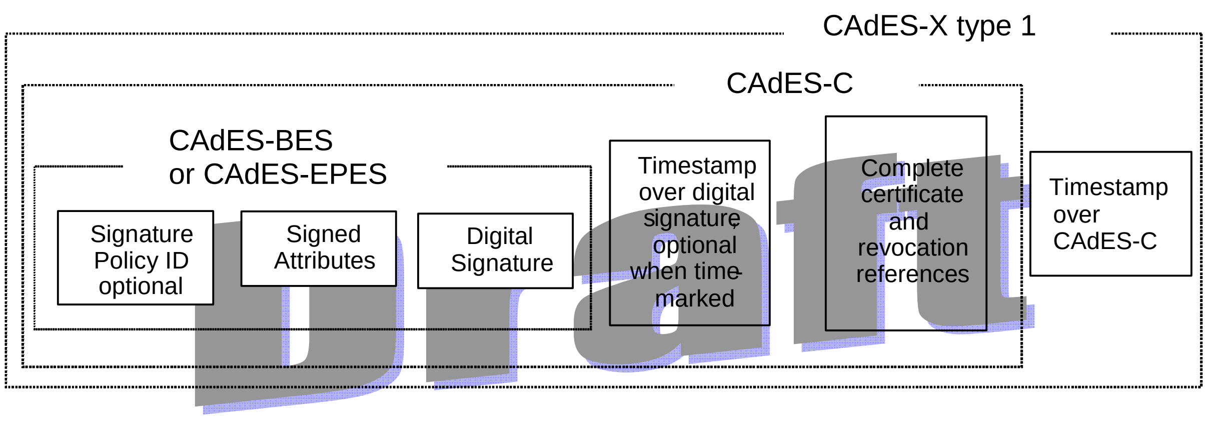

CAdES-X type 1
CAdES-C
CAdES-BES or CAdES-EPES
Signature Policy ID optional
Signed Attributes
Digital Signature
Timestamp over digital signature optional when time-marked
Complete certificate and revocation references
Timestamp over CAdES-C

Figure B.2: Illustration of CAdES-X Type 1

B.2.2 CAdES-X Type 2

Extended format with time type 2 (**CAdES-X Type 2**), in accordance with the present document, is illustrated in figure B.3. It adds to the CAdES-C signature one or more `CAdES-C-time-stamped-certs-crls-references` attributes. Clause A.1.5.1 provides the specification details on the `CAdES-C-time-stamped-certs-crls-references` attribute.

This form represents an AdES-X Type 2 signature as described in ETSI EN 319 102 [i.5].

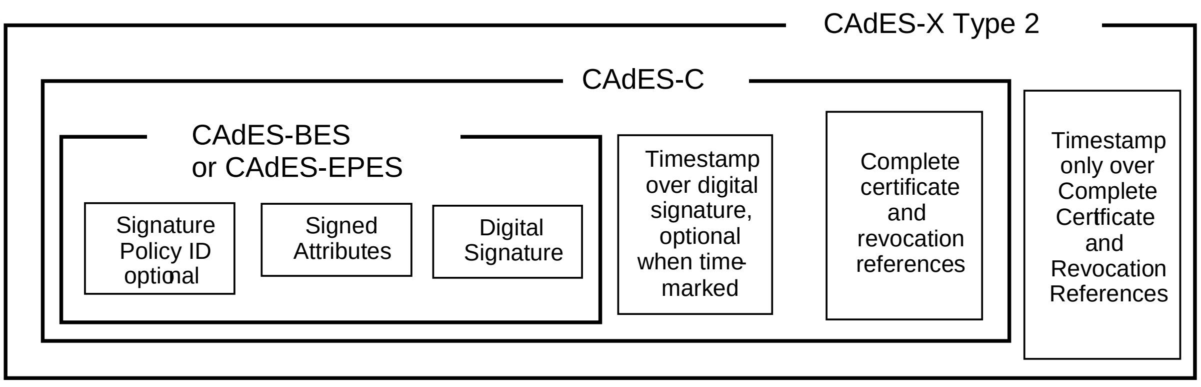

CAdES-X Type 2
CAdES-C
CAdES-BES or CAdES-EPES
Signature Policy ID optional
Signed Attributes
Digital Signature
Timestamp over digital signature, optional when time-marked
Complete certificate and revocation references
Timestamp only over Complete Certficate and Revocation References

Figure B.3: Illustration of CAdES-X Type 2

B.3 Extended Long electronic signatures without and with time forms

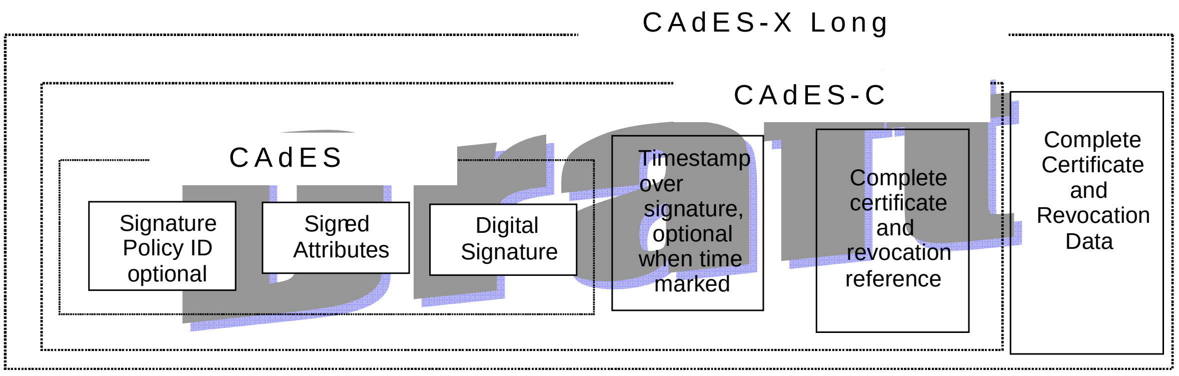
B.3.1 Extended Long electronic signature (CAdES-X-L)

Extended Long format (**CAdES-X Long**), in accordance with the present document, adds the `certificate-values` and `revocation-values` attributes to the CAdES-C format Clauses A.1.1.2 and A.1.2.2 give specification details on these attributes.

An electronic signature with the additional validation data forming the CAdES-X Long form (CAdES-X-Long) is illustrated in figure B.4. It is build upon CAdES-C.

The following attributes are required if the full certificate values and revocation values are not already included in the CAdES-BES or CAdES-EPES:

- `certificate-values` attribute, `as defined in clause` A.1.1.2;
- `revocation-values` attribute, `as defined in clause` A.1.2.2.


CAdES-X Long
CAdES-C
CAdES
Signature Policy ID optional
Signed Attributes
Digital Signature
Timestamp over signature, optional when time marked
Complete certificate and revocation reference
Complete Certificate and Revocation Data
Draft

NOTE: Time-stamp tokens may themselves include unsigned attributes required to validate the time-stamp token.

Figure B.4: Illustration of CAdES-X-Long

B.3.2 Extended Long electronic signatures with time forms (CAdES-X-L Type 1 or 2)

Extended Long with Time (**CAdES-X Long Type 1 or 2**), in accordance with the present document, is a combination of **CAdES-X Long** and one of the two forms described in clause B.2 (**CAdES-X Type 1** and **CAdES-X Type 2**).

An electronic signature with the additional validation data forming the **CAdES-X Long Type 1** and **CAdES-X Long Type 2** is illustrated in figure B.5.

This form represents an AdES-X-L Type signature as described in ETSI EN 319 102 [i.5].

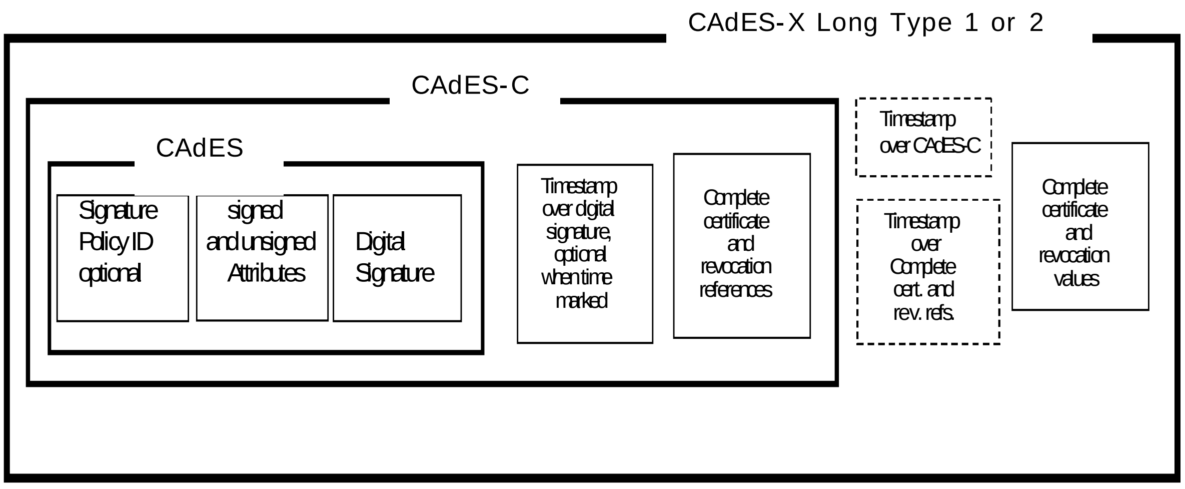

CAdES-X Long Type 1 or 2
CAdES-C
CAdES
Signature Policy ID optional
signed and unsigned Attributes
Digital Signature
Timestamp over digital signature, optional when time marked
Complete certificate and revocation references
Timestamp over CAdES-C
Timestamp over Complete cert. and rev. refs.
Complete certificate and revocation values

Figure B.5: Illustration of CAdES-X Long Type 1 and CAdES-X Long Type 2

B.4 CAdES-A with archive-time-stamp (ATSv2) attribute (obsolete)

Archival Form (CAdES-A) with `archive-time-stamp` (ATSv2), in accordance with the present document builds on a **CAdES-X Long** or a **CAdES-X Long Type 1 or 2** by adding one or more `archive-time-stamp` attributes (clause A.2.2). This form was used for archival of long-term signatures. Successive time-stamps protect the whole material against vulnerable hashing algorithms or the breaking of the cryptographic material or algorithms.

This form is kept for backwards compatibility.

Several instances of the `archive-time-stamp` attribute may occur with an electronic signature, both over time and from different TSUs. The time-stamp should be created using stronger algorithms (or longer key lengths) than in the original electronic signatures or time-stamps.

This form represents an AdES-A signature as described in ETSI EN 319 102 [i.5].

The structure of the CAdES-A form is illustrated in figure B.6.

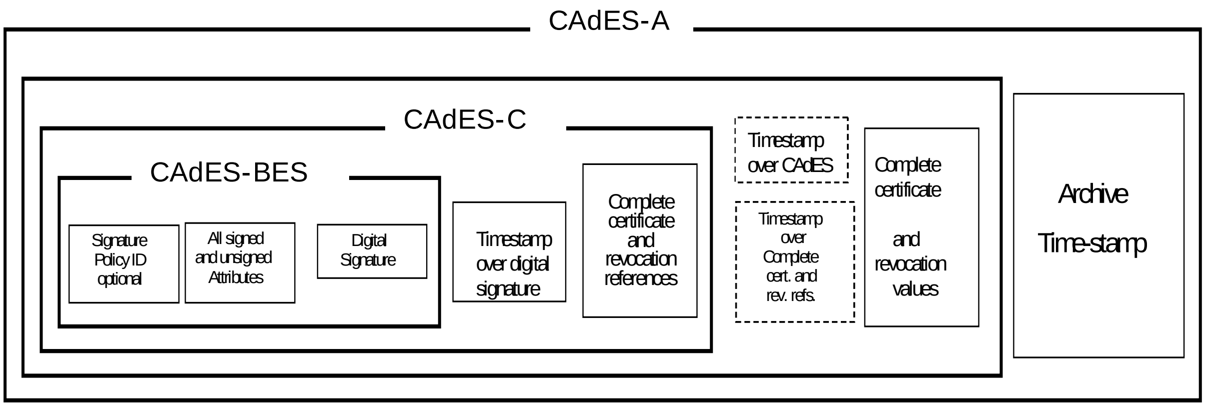

CAdES-A
CAdES-C
CAdES-BES
Signature Policy ID optional
All signed and unsigned Attributes
Digital Signature
Timestamp over digital signature
Complete certificate and revocation references
Timestamp over CAdES
Timestamp over Complete cert. and rev. refs.
Complete certificate and revocation values
Archive Time-stamp

Figure B.6: Illustration of CAdES-A

B.5 Long-Term Electronic Signature (CAdES-LT) (obsolete)

Long-Term Form (CAdES-LT), in accordance with the present document builds on any format among **CAdES-T, CAdES-C, CAdES-X Long, CAdES-X Long Type 1 or 2** or a **CAdES-A** by adding one or more `long-term-validation` attributes. This form is used for archival of long-term signatures. CAdES-LT and CAdES-A provide alternative solutions to the same problem

NOTE: This attribute is very similar in spirit to the CAdES-X Long and CAdES-A forms. It is however simpler to implement and more flexible than these, and contains a POE feature that is not provided by CAdES-X-L.

The use of the CAdES-LT form is deprecated for any signature to which a long-term-validation attribute has not already been applied.

The CAdES-LT is created by adding to CAdES-T, CAdES-C, CAdES-X, CAdES-X-L, CAdES-X-L Type 1 or 2, or CAdES-A the following element:

- a `long-term-validation` attribute, defined in clause A.2.3.

Several instances of the `long-term-validation` attribute may occur with an electronic signature, both over time and from different TSUs. The time-stamp should be created using stronger algorithms (or longer key lengths) than in the original electronic signatures or time-stamps.

An example of a structure of a CAdES-LT form is illustrated in figure B.7.

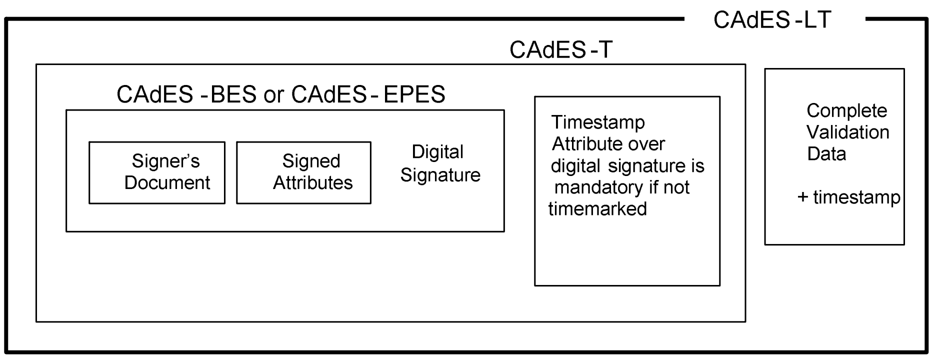


Figure B.7: Illustration of CAdES-LT

Annex C (normative): Conformance requirment for additional Electronic Signature formats

In addition to the base conformance levels defined in clause 7, we introduce in the following additional core conformance levels:

- Electronic signatures with complete validation data references (CAdES-C)
- Extended siganturew with time forms, Type 1 (CAdES-X Type 1)
- Extended siganturew with time forms, Type 2 (CAdES-X Type 2)
- eXtended long exlectronic signature (CAdES-X-L)
- eXtended long electronic signatures with time forms, Type 1 (CAdES-X-L Type 1)
- eXtended long electronic signatures with time forms, Type 2 (CAdES-X-L Type 2)

An implementation claiming to be conformant to one of them shall fulfil the corresponding requirements defined in the current annex.

C.1 Electronic signatures with complete validation data references (CAdES-C)

A signature corresponding to CAdES-C level shall be build upon a signature having CAdES-T level. In addition, it shall contain the following unsigned attributes:

- `complete-certificate-references` (see clause A.1.1.1)
- `complete-revocation-references` (see clause A.1.2.1)

If attributes certificates are used, then the following attributes may be present:

- `attribute-certificate-references` (see clause A.1.3)
- `attribute-revocation-references` (see clause A.1.4)

C.2 eXtended signatures with time forms (CAdES-X)

The extended signatures with time forms build on CAdES-C and protect the certificate and revocation references by a time-stamp.

C.2.1 CAdES-X Type 1

A signature corresponding to CAdES-X Type 1 level shall be build upon a signature having CAdES-C level. In addition, it shall contain the following unsigned attribute:

- `CAdES-C-Timestamp` attribute, as defined in clause A.1.5.2

C.2.2 CAdES-X Type 2

A signature corresponding to CAdES-X Type 2 level shall be build upon a signature having CAdES-C level. In addition, it shall contain the following unsigned attribute:

- `time-stamped-certs-crls-references` attribute, as defined in clause A.1.5.1

C.3 eXtended long electronic signature with and without time

Extended long electronic signatures adds the certificate and revocation data to the signature.

C.3.1 eXtended long electronic signature (CAdES-X-L)

A signature corresponding to CAdES-X-L level shall be build upon a signature having CAdES-C level. In addition, if the full certificate values and revocation values are not already included in the CAdES-BES or CAdES-EPES, the signature shall contain the following

- `certificate-values` attribute, as defined in clause A.1.1.2;
- `revocation-values` attribute, as defined in clause A.1.2.2.

C.3.2 eXtended long electronic signatures with time forms

C.3.2.1 eXtended long electronic signatures with time forms, Type 1 (CAdES-X-L Type 1)

A signature corresponding to CAdES-X-L Type 1 level shall be build upon a signature having CAdES-X Type 1 level. In addition, if the full certificate values and revocation values are not already included in the CAdES-BES or CAdES-EPES, the signature shall contain the following

- `certificate-values` attribute, as defined in clause A.1.1.2;
- `revocation-values` attribute, as defined in clause A.1.2.2.

C.3.2.1 eXtended long electronic signatures with time forms, Type 2 (CAdES-X-L Type 2)

A signature corresponding to CAdES-X-L Type 2 level shall be build upon a signature having CAdES-X Type 2 level. In addition, if the full certificate values and revocation values are not already included in the CAdES-BES or CAdES-EPES, the signature shall contain the following

- `certificate-values` attribute, as defined in clause A.1.1.2;
- `revocation-values` attribute, as defined in clause A.1.2.2.

Annex D (informative): Attributes of the Electronic Signature

This annex gives some rationales about the attributes used in the electronic signature.

D.1 The `signing-time` attribute

The present document provides the capability to include a claimed signing time as an attribute of an electronic signature. Readers are reminded that this is not a time indication coming from a trusted source like a time-stamp token.

D.2 Signing certificate reference attributes

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 citizen of a nation 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 that multiple private keys are used, 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 vulnerable to substitution attacks. If the certificate from the signer was simply appended to the signature and thus not protected by the signature, anyone could substitute one certificate for another, and the message would appear to be signed by someone else. In order to counter this kind of attack, the identifier of the signer has to be protected by the digital signature from the signer.

The `signing-certificate` and `signing-certificate-v2` are designed to prevent the simple substitution of the certificate.

D.3 The `commitment-type-indication` attribute

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.

If the indicated commitment type is explicit using 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 type indications may be subject to signer and verifier agreement, specified as part of the signature policy, or registered for generic use across multiple policies.

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 an identifier (OID) and an optional sequence of qualifiers, which may provide additional information (for instance information about the context, be it contractual/legal/application specific).

The signature policy specifies a set of attributes that it "recognizes". This "recognized" set includes all those commitment types defined as part of the signature policy, as well as any externally defined commitment types that the policy may choose to recognize. Only recognized commitment types are allowed in the `commitment-type` attribute.

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

NOTE: Examples of commitment indicated through the semantics of the data being signed are:

- an explicit commitment made by the signer indicated by the type of data being signed over. Thus, the data structure being signed can have an explicit commitment within the context of the application (e.g. EDIFACT purchase order);
- an implicit commitment that is a commitment made by the signer because the data being signed over has specific semantics (meaning), which is only interpretable by humans (i.e. free text).

D.4 Attributes for identifying the signed data type

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, further typing information should be used to identify the type of document being signed. This is generally achieved using the MIME content typing and encoding mechanism defined in RFC 2045 [1]). Further information on the use of MIME is given in annex F. 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

D.4.1 The `content-hints` attribute

The `contents-hints` attribute provides information on the innermost signed content of a multi-layer message where one content is encapsulated in another. This may be useful if the signed data is itself encrypted.

D.4.2 The `mime-type` attribute

The `mime-type` attribute provides information on the mime-type of the signed data.

D.5 The `signer-location` attribute

In some transactions, the purported location of the signer at the time he or she applies his signature may need to be indicated. In order to provide this information the optional `signer-location` signed attribute is defined. It specifies an address associated with the signer at a particular geographical (e.g. city) location.

D.6 Attributes of the signer (Role)

While the name of the signer is important, the position of the signer within a company or an organization is of paramount importance as well. Some information (i.e. a contract) 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 three different ways for providing this feature:

- by placing a *claimed* signer attribute / role in the signed CMS attributes field;

- by placing a an assertion that includes signer attribute / role, signed by an entity which does not satisfy the requirements needed to be considered an Attribute Authority; or
- by placing an attribute certificate issued by an Attribute Authority.

NOTE: Another possible approach would have been to use additional attributes containing the attribute or roles name(s) in the signer's identity certificate. However, it was decided not to follow this approach as it significantly complicates the management of certificates. For example, by using separate certificates for the signer's identity and roles means new identity keys need not be issued if a user's role changes.

D.6.1 Claimed signer attribute/role

The signer may be trusted to state his own attribute or role without any third party certifying this claim; in which case, the claimed role can be added to the signature as a signed attribute.

D.6.2 Certified signer attribute/role

A certified signer attribute / role is certified by a trusted Attribute Authority (AA). The certification is done either by the AA issuing an Attribute Certificate or by the AA signing a SAML assertion

Unlike public key certificates that bind an identifier to a public key, Attribute Certificates bind the identifier of a certificate to some attributes, like a role. An Attribute Certificate is NOT issued by a CA but by an Attribute Authority (AA). The Attribute Authority, in most cases, might be 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 be 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. In order to do so, the Attribute Certificate will have to be included in the signed data in order to be protected by the digital signature from the signer. In order to unambiguously identify the attribute certificate(s) to be used for the verification of the signature, an identifier of the attribute certificate(s) from the signer is part of the signed data.

D.6.3 Signed assertion

A signed assertion is stronger than a claimed assertion but less strict than an attribute certificate. A (trusted) third party asserts specific attributes/roles of the signer and commits to this assertion by signing it.

D.7 Multiple signatures and countersignatures

Some electronic signatures may only be valid if they bear more than one signature. This is generally the case 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. Multiple independent signatures are supported by independent `SignerInfo` from each signer. The computation of these signatures is performed on exactly the same input but using different private keys.

Independent signatures may be used to provide independent signatures from different parties with different signed attributes, or to provide multiple signatures from the same party using alternative signature algorithms, in which case the other attributes, excluding time values and signature policy information, will generally be the same.

Countersignatures are applied one after the other and are used where the order in which the signatures are applied is important Multiple embedded signatures are supported using the `countersignature` unsigned attribute (see clause 6.2.7). Each counter signature is carried in `countersignature` held as an unsigned attribute to the `SignerInfo` to which the counter-signature is applied.

Counter signatures may be used to provide signatures from different parties with different signed attributes, or to provide multiple signatures from the same party using alternative signature algorithms, in which case the other attributes, excluding time values and signature policy information, will generally be the same.

All other multiple signature schemes, e.g. a signed document with a countersignature, double countersignatures, or multiple signatures, can be reduced to one or more occurrences of the above two cases.

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, which is out of scope of CAdES. Readers are reminded, though, that ASiC containers specify a standard way of packaging together documents and their detached signatures (be them independent or countersignatures).

D.8 The `content-time-stamp` attribute

The `content-time-stamp` attribute is an optional attribute that allows to time-stamp the content and to provide proof of the existence of the content, at the time indicated by the time-stamp token.

Using this optional attribute, a trusted secure time may be obtained before the document is signed and included under the digital signature. This solution requires an online connection to a trusted time-stamping service before generating the signature and may not represent the precise signing time, since it can be obtained in advance. However, this optional attribute may be used by the signer to prove that the signed object existed before the date included in the time-stamp (see clause 6.2.8).

D.9 The `signature-policy-identifier` attribute

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. A *signature policy* may be issued, for example, by a party relying on the electronic signatures and selected by the signer for use with that relying party. Alternatively, a signature policy may be established through an electronic trading association for use amongst its members. Both the signer and verifier use the same signature policy.

The signature policy may be explicitly identified or may be implied by the semantics of the data being signed and other external data, like a contract being referenced, which itself refers to a signature policy.

An explicit signature policy has a globally unique reference, which is bound to an electronic signature by the signer as part of the signature calculation.

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 comprehensively defined and in a computer-processable form.

The signature policy thus includes the following:

- rules that apply to technical validation of a particular signature;
- rules that may be implied through adoption of Certificate Policies that apply to the electronic signature (e.g. rules for ensuring the secrecy of the private signing key);
- rules that relate to the environment used by the signer, e.g. the use of an agreed CAD (Card Accepting Device) used in conjunction with a smart card.

For example, the major rules required for technical validation can include:

- recognized root keys or "top-level certification authorities";
- acceptable certificate policies (if any);

- necessary certificate extensions and values (if any);
- the need for the revocation status for each component of the certification tree;
- acceptable TSAs (if time-stamp tokens are being used);
- acceptable organizations for keeping the audit trails with time-marks (if time-marking is being used);
- acceptable AAs (if any are being used); and
- rules defining the components of the electronic signature that have to be provided by the signer with data required by the verifier when required to provide long-term proof.

For a detailed discussion on signature policies see EN 319 172-1 [i.8].

When two independent parties want to evaluate an electronic signature, it is fundamental that they get the same result. This requirement can be met using comprehensive signature policies that ensure consistency of signature validation. Signature policies can be identified implicitly by the data being signed, or they can be explicitly identified using the CAdES-EPES form of electronic signature; the CAdES-EPES mandates a consistent signature policy to be used by both the signer and verifier.

By signing over the Signature Policy Identifier in the CAdES-EPES, the signer explicitly indicates that he or she has applied the signature policy in creating the signature.

In order to unambiguously identify the details of an explicit signature policy that is to be used to verify a CAdES-EPES, the signature, an identifier, and hash of the "Signature policy" is part of the signed data. Additional information about the explicit policy (e.g. web reference to the document) may be carried as "qualifiers" to the Signature Policy Identifier.

In order to unambiguously identify the authority responsible for defining an explicit signature policy, the "Signature policy" can be signed.

D.10 The `signature-policy-store` attribute

In some cases, the validator would like to be able to do signature validations off-line. Or the archival service wants to make sure that the data necessary for validating a signature is available after a very long period. In that case, the validator might add the encoded signature policy or a link to a local storage of the signature policy into the signature using the `signature-policy-store` attribute. However, it is the responsibility of the party adding the `signature-policy-store` to make sure that the right signature policy is added.

D.11 Content Cross-Referencing

When presenting a signed data is in relation to another signed data, it may be important to identify the signed data to which it relates. The `content-reference` and `content-identifier` attributes, as defined in ESS (RFC 2634 [2]), provide the ability to link a request and reply messages in an exchange between two parties.

D.12 The `signature-time-stamp` attribute

An important property for long standing signatures is that a signature, having been found once to be valid, will continue to be so months or years later. A signer, verifier or both may be required to provide on request, proof that an electronic signature was created or validated 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 validated.

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 time-stamp is NOT a component of the Basic Electronic Signature, but it is the essential component of the electronic signature with Time.

It is required, in the present document, that if a signer's digital signature value is to be time-stamped, the time-stamp token is issued by a trusted source, known as a Time-Stamping Authority.

The validation mandated by the signature policy can specify a maximum acceptable time difference which is allowed between the time indicated in the `signing-time` element and the time indicated by the `signature-time-stamp` attribute.

D.13 Elements containing references to validation data

When dealing with long term electronic signatures, all the data used in the verification (namely, certificate path and revocation information) of such signatures have to 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 attributes) for archival purposes (see archival electronic signatures CAdES-A).

Systems implementing the present document may alternatively consider to archive validation data outside the CAdES e.g. to prevent redundant storage and to reduce the size of the signatures. In such cases each electronic signature incorporates references to all these data within the signature, reducing accordingly the size of the stored electronic signature. This format builds up taking CAdES-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 that have been used to validate the attribute certificate(s), if present;
- the references to the full set of revocation data that have been used in the validation of the attribute certificate(s), if present.

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. In consequence, means are provided in the present document for referencing both types of revocation data.

D.14 Time-stamps on references to validation data

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

- Time-stamp the sequence formed by the digital signature (`SignatureValue` element), the `signature-time-stamp` element when present in the CAdES-T form, the certification path references, the Attribute Authorities references and the revocation data references (for both the certificates in the certification path and in the list of Attribute Authorities certificate. This form **(CAdES-X Type 1**) is suitable to be used with an OCSP response, and it offers the additional advantage of providing an integrity protection over the whole data.

- Time-stamp only the references. This form (**CAdES-X Type2**) is suitable to be used with CRLs, since the time-stamped information may be used for more than one signature (when signers have their certificates issued by the same CA and when signatures can be checked using the same CRLs).

For both ways signer, verifier or both may request, obtain and add the time-stamp to the electronic signature.

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 (`SignatureValue` element), the signature time-stamp(s) present in the CAdES-T form, the certification path references and the revocation status references (by adding a `CAdES-C-Timestamp` to the electronic signature). 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.

Despite the fact that this is the best scenario for using a time-stamp that covers both the references and the signature elements, it may also be used in scenarios where the revocation data are CRLs. However, time-Stamping each electronic signature with the complete validation data references 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 references to commonly used certificates and CRLs (the time-stamp token that `CAdES-C-time-stamped-certs-crls-references` unsigned property encapsulates), 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). As said before, the information that needs to be time-stamped is not the actual certificates and CRLs but the unambiguous references to those certificates and CRLs.

D.15 Validation data

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. Also the certificates used for validating any attribute certificate and/or time-stamp present within the electronic signature.

When dealing with long term electronic signatures, all the data used in the validation (including the certification paths of the signing certificate, and any attribute certificate and/or time-stamp present) have to be conveniently stored and time-stamped.

Several elements within a CAdES signature are entitled to contain certificate values that have been used to validate it, namely: `SignedData.certificates, certificate-values and long-term-validation.extraCertificates.`

The `SignedData.certificates` allows to store the certificates in a global store. This can be helpful to avoid that certificates are stored several times if they are used in the validation of different signatures. When using archive-timestamp-v3 the certificates should be stored in `SignedData.certificates.` This can only be done if no `archive-timestamp` or `long-term-validation` attribute was added.

The `certificate-values` attribute allows to store all the certificates that are not yet covered by `SignedData.certificates` in an unsigned attribute specific to the signature.

The `extraCertificates` field is part of the obsolete `long-term-validation` attribute. It should only be used to extend the lifetime of signature containing already `long-term-validation` attributes.

For dealing with long term signatures, it is also needed to store and conveniently time-stamp all the revocation data used in the validation of such signatures.

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. Usually this is done as soon as possible, after the grace period, to minimize the time delay between the generation and validation 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 will be conveyed to the verifier. Additional CRLs for the CA certificates in the certificate path need to 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". Usually this is done as soon as possible after the generation of the signature, after the grace period. 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.

Several elements within a CAdES signature are entitled to contain revocation data values that have been used to validate it, namely: `SignedData.crls, revocation-values and long-term-validation.extraRevocation`.

The `SignedData.crls` allows to store revocation data values in a global store. This can be helpful to avoid that revocation data values like CRLs are stored several times if they are used in the validation of different signatures. When using archive-timestamp-v3 the revocation data values should be stored in `SignedData.crls.` This can only be done if no `archive-timestamp` or `long-term-validation` attribute was added.

The `revocation-values` attribute allows to store all the revocation data values that are not yet covered by `SignedData.crls` in an unsigned attribute specific to the signature.

The `extraRevocation` field is part of the obsolete `long-term-validation` attribute. It should only be used to extend the lifetime of signature containing already `long-term-validation` attributes.

D.16 Time-stamps for archival

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 cryptanalytical 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, the present document specifies the Archive validation data technique, covering all the cases.

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 Trust 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.

This technique is referred to as **CAdES-A** in the present document.

Annex E (normative): ASN.1 Definitions

This annex provides a summary of all the ASN.1 syntax definitions for new syntax defined in the present document.

E.1 Signature Format Definitions Using X.208 ASN.1 Syntax

NOTE: The ASN.1 module defined in clause E.1 using syntax defined in Recommendation ITU-T X.208 [10] has precedence over that defined in clause E.2 in the case of any conflict.

```
ETS-ElectronicSignatureFormats-ExplicitSyntax88 { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-mod(0) eSignature-explicit88(28)}

DEFINITIONS EXPLICIT TAGS ::=
BEGIN
-- EXPORTS All

IMPORTS

-- Cryptographic Message Syntax (CMS): RFC 5652
    ContentInfo, ContentType, id-data, id-signedData, SignedData, EncapsulatedContentInfo,
    SignerInfo, id-contentType, id-messageDigest, MessageDigest, id-signingTime, SigningTime,
    id-countersignature, Countersignature
    FROM CryptographicMessageSyntax2004
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) cms-2004(24) }


-- ESS Defined attributes: ESS Update
-- RFC 5035 (Adding CertID Algorithm Agility)

   id-aa-signingCertificate, SigningCertificate, IssuerSerial,
    id-aa-contentReference, ContentReference, id-aa-contentIdentifier, ContentIdentifier,
    id-aa-signingCertificateV2
    FROM ExtendedSecurityServices-2006
        { iso(1) member-body(2) us(840) rsadsi(113549)
          pkcs(1) pkcs-9(9) smime(16) modules(0) id-mod-ess-2006(30) }


-- Internet X.509 Public Key Infrastructure - Certificate and CRL Profile: RFC 5280

   Certificate, AlgorithmIdentifier, CertificateList, Name,
   DirectoryString, Attribute, BMPString, UTF8String
      FROM PKIX1Explicit88
   {iso(1) identified-organization(3) dod(6) internet(1)
   security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-explicit (18)}

   GeneralNames, GeneralName, PolicyInformation
      FROM PKIX1Implicit88
   {iso(1) identified-organization(3) dod(6) internet(1)
   security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-implicit (19)}

-- Internet Attribute Certificate Profile for Authorization: RFC 3281
   AttributeCertificate
      FROM PKIXAttributeCertificate
      {iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-attribute-cert(12)}

-- OCSP RFC 6960
   BasicOCSPResponse, ResponderID
    FROM OCSP {iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-ocsp(14)}
```

```
-- Time Stamp Protocol RFC 3161
TimeStampToken
    FROM PKIXTSP
    {iso(1) identified-organization(3) dod(6) internet(1)    security(5)
    mechanisms(5) pkix(7) id-mod(0) id-mod-tsp(13)}

-- Evidence Record Syntax RFC 4998
EvidenceRecord
    FROM ERS
    {iso(1) identified-organization(3) dod(6) internet(1) security(5)
    mechanisms(5) ltans(11) id-mod(0) id-mod-ers88(2) id-mod-ers88-v1(1)}
;


-- Definitions of Object Identifier arcs used in the present document
-- ==================================================================

-- OID used referencing electronic signature mechanisms based on the present document
-- for use with the Independent Data Unit Protection (IDUP) API (see Annex D)

id-etsi-es-IDUP-Mechanism-v1 OBJECT IDENTIFIER ::=
    { itu-t(0) identified-organization(4) etsi(0)
       electronic-signature-standard (1733) part1 (1) idupMechanism (4) etsiESv1(1) }

id-etsi-es-attributes OBJECT IDENTIFIER ::=
    { itu-t(0) identified-organization(4) etsi(0)
       electronic-signature-standard (1733) attributes(2) }


-- Basic ES CMS Attributes Defined in the present document
-- =======================================================

-- OtherSigningCertificate - deprecated

id-aa-ets-otherSigCert OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-aa(2) 19 }

OtherSigningCertificate ::=  SEQUENCE {
    certs         SEQUENCE OF OtherCertID,
    policies      SEQUENCE OF PolicyInformation OPTIONAL
                  -- NOT USED IN THE PRESENT DOCUMENT
    }

OtherCertID ::= SEQUENCE {
     otherCertHash            OtherHash,
     issuerSerial             IssuerSerial OPTIONAL }

OtherHash ::= CHOICE {
    sha1Hash OtherHashValue,  -- This contains a SHA-1 hash
    otherHash OtherHashAlgAndValue}


-- Policy ES Attributes Defined in the present document
-- ==================================================

-- Mandatory Basic Electronic Signature Attributes as above, plus in addition.


-- Signature-policy-identifier attribute

id-aa-ets-sigPolicyId OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-aa(2) 15 }


SignaturePolicyIdentifier  ::= CHOICE {
         signaturePolicyId          SignaturePolicyId,
         signaturePolicyImplied     SignaturePolicyImplied
                                    -- not used in this version
}
```

```
SignaturePolicyId ::= SEQUENCE {
         sigPolicyId           SigPolicyId,
         sigPolicyHash         SigPolicyHash,
         sigPolicyQualifiers   SEQUENCE SIZE (1..MAX) OF
                                   SigPolicyQualifierInfo OPTIONAL
}

 SignaturePolicyImplied ::= NULL


SigPolicyId ::= OBJECT IDENTIFIER

SigPolicyHash ::= OtherHashAlgAndValue

   OtherHashAlgAndValue ::= SEQUENCE {
      hashAlgorithm   AlgorithmIdentifier,
      hashValue       OtherHashValue }

   OtherHashValue ::= OCTET STRING

SigPolicyQualifierInfo ::= SEQUENCE {
        sigPolicyQualifierId  SigPolicyQualifierId,
        sigQualifier          ANY DEFINED BY sigPolicyQualifierId }

SigPolicyQualifierId ::=
        OBJECT IDENTIFIER

    id-spq-ets-uri OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-spq(5) 1 }


SPuri ::= IA5String

    id-spq-ets-unotice OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-spq(5) 2 }

   SPUserNotice ::= SEQUENCE {
        noticeRef        NoticeReference OPTIONAL,
        explicitText     DisplayText OPTIONAL}

   NoticeReference ::= SEQUENCE {
        organization     DisplayText,
        noticeNumbers    SEQUENCE OF INTEGER }

   DisplayText ::= CHOICE {
        visibleString    VisibleString  (SIZE (1..200)),
        bmpString        BMPString      (SIZE (1..200)),
        utf8String       UTF8String     (SIZE (1..200)) }

SPDocSpecification ::= CHOICE {
        oid     OBJECT IDENTIFIER,
        uri     IA5String }

    id-spq-ets-docspec OBJECT IDENTIFIER ::=  xxxx



-- Optional Electronic Signature Attributes

-- Signature-policy-store attribute
id-aa-ets-sigPolicyStore OBJECT IDENTIFIER ::= xxxx

SignaturePolicyStore ::=SEQUENCE {
        spDocSpec      SPDocSpecification ,
        spDocument     SignaturePolicyDocument }

SignaturePolicyDocument ::= CHOICE {
        sigPolicyEncoded      OCTET STRING,
        sigPolicyLocalURI     IA5String }


-- Commitment-type attribute

id-aa-ets-commitmentType OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 16}
```

```
CommitmentTypeIndication ::= SEQUENCE {
  commitmentTypeId CommitmentTypeIdentifier,
  commitmentTypeQualifier SEQUENCE SIZE (1..MAX) OF CommitmentTypeQualifier OPTIONAL}

CommitmentTypeIdentifier ::= OBJECT IDENTIFIER

CommitmentTypeQualifier ::= SEQUENCE {
        commitmentTypeIdentifier CommitmentTypeIdentifier,
        qualifier   ANY DEFINED BY commitmentTypeIdentifier }


    id-cti-ets-proofOfOrigin OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 1}

    id-cti-ets-proofOfReceipt OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 2}

    id-cti-ets-proofOfDelivery OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 3}

    id-cti-ets-proofOfSender OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 4}

    id-cti-ets-proofOfApproval OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 5}

    id-cti-ets-proofOfCreation OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 6}


-- Signer-location attribute

id-aa-ets-signerLocation OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 17}


SignerLocation ::= SEQUENCE { -- at least one of the following shall be present
            countryName [0] DirectoryString OPTIONAL,
                -- As used to name a Country in X.500
        localityName [1] DirectoryString OPTIONAL,
                -- As used to name a locality in X.500
            postalAdddress [2] PostalAddress OPTIONAL }

    PostalAddress ::= SEQUENCE SIZE(1..6) OF DirectoryString


-- Signer-attributes attribute

id-aa-ets-signerAttr OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 18}


SignerAttribute ::= SEQUENCE OF CHOICE {
            claimedAttributes   [0] ClaimedAttributes,
            certifiedAttributes [1] CertifiedAttributes }

ClaimedAttributes ::= SEQUENCE OF Attribute

CertifiedAttributes ::= AttributeCertificate  -- as defined in RFC 3281: see clause 4.1

-- Signer-attributes-v2 attribute

id-aa-ets-signerAttrV2 OBJECT IDENTIFIER ::= xxxx

SignerAttributeV2 ::= SEQUENCE {
            claimedAttributes       [0] ClaimedAttributes OPTIONAL,
            certifiedAttributesV2   [1] CertifiedAttributesV2 OPTIONAL,
            signedAssertions        [2] SignedAssertions OPTIONAL}

ClaimedAttributes ::= SEQUENCE OF Attribute

CertifiedAttributesV2 ::= SEQUENCE OF CHOICE {
            attributeCertificate        [0] AttributeCertificate,
                                            -- as defined in RFC 3281: see clause 4.1.
            otherAttributeCertificate   [1] OtherAttributeCertificate }

OtherAttributeCertificate ::= SEQUENCE {
        otherAttributeCertID        OtherAttributeCertID,
        otherAttributeCert          ANY DEFINED BY otherAttributeCertID }

OtherAttributeCertID ::= OBJECT IDENTIFIER

SignedAssertions ::= SEQUENCE OF SignedAssertion

SignedAssertion ::= SEQUENCE {
        signedAssertionID       SignedAssertionID,
        signedAssertion         ANY DEFINED BY signedAssertionID }

SignedAssertionID ::= OBJECT IDENTIFIER

-- claimed-SAML-assertion attribute

id-aa-ets-claimedSAML OBJECT IDENTIFIER ::= xxxx

ClaimedSAMLAssertion ::= OCTET STRING.



-- Content-timestamp attribute

id-aa-ets-contentTimestamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 20}


ContentTimestamp::= TimeStampToken

-- Mime type

id-aa-ets-mimeType OBJECT IDENTIFIER ::= { itu-t(0) identified-organization(4) etsi(0)
       electronic-signature-standard (1733) attributes(2) 1}

MimeType::= UTF8String

-- Signature-timestamp attribute

id-aa-signatureTimeStampToken OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 14}

SignatureTimeStampToken ::= TimeStampToken

-- Complete-certificate-references attribute

id-aa-ets-certificateRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 21}


CompleteCertificateRefs ::=  SEQUENCE OF OtherCertID


-- Complete-revocation-references attribute

id-aa-ets-revocationRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 22}

CompleteRevocationRefs ::=  SEQUENCE OF CrlOcspRef

CrlOcspRef ::= SEQUENCE {
    crlids          [0] CRLListID   OPTIONAL,
    ocspids         [1] OcspListID  OPTIONAL,
    otherRev        [2] OtherRevRefs OPTIONAL
}

CRLListID ::=  SEQUENCE {
    crls        SEQUENCE OF CrlValidatedID}

CrlValidatedID ::=  SEQUENCE {
     crlHash                    OtherHash,
     crlIdentifier              CrlIdentifier OPTIONAL}

CrlIdentifier ::= SEQUENCE {
    crlissuer                 Name,
    crlIssuedTime             UTCTime,
    crlNumber                 INTEGER OPTIONAL
}
```

```
OcspListID ::=  SEQUENCE {
    ocspResponses        SEQUENCE OF OcspResponsesID}

OcspResponsesID ::=  SEQUENCE {
    ocspIdentifier              OcspIdentifier,
    ocspRefHash                 OtherHash    OPTIONAL
}

OcspIdentifier ::= SEQUENCE {
    ocspResponderID     ResponderID,    -- As in OCSP response data
    producedAt          GeneralizedTime -- As in OCSP response data
}

OtherRevRefs ::= SEQUENCE {
    otherRevRefType OtherRevRefType,
    otherRevRefs    ANY DEFINED BY otherRevRefType
 }

OtherRevRefType ::= OBJECT IDENTIFIER

-- Certificate-values attribute

id-aa-ets-certValues OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 23}

CertificateValues ::=  SEQUENCE OF Certificate

-- Certificate-revocation-values attribute

id-aa-ets-revocationValues OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 24}

RevocationValues ::=  SEQUENCE {
   crlVals          [0] SEQUENCE OF CertificateList OPTIONAL,
   ocspVals         [1] SEQUENCE OF BasicOCSPResponse OPTIONAL,
   otherRevVals     [2] OtherRevVals OPTIONAL}

OtherRevVals ::= SEQUENCE {
    otherRevValType OtherRevValType,
    otherRevVals    ANY DEFINED BY otherRevValType
 }

OtherRevValType ::= OBJECT IDENTIFIER


-- CAdES-C time-stamp attribute

id-aa-ets-escTimeStamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 25}

ESCTimeStampToken ::= TimeStampToken


-- Time-Stamped Certificates and CRLs

id-aa-ets-certCRLTimestamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 26}

TimestampedCertsCRLs ::= TimeStampToken


-- Archive time-stamp attribute

id-aa-ets-archiveTimestampV2 OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 48}

ArchiveTimeStampToken ::= TimeStampToken

-- archive-time-stamp-v3 attribute
id-aa-ets-archiveTimestampV3 OBJECT IDENTIFIER ::= { itu-t(0)
  identified-organization(4) etsi(0)
  electronic-signature-standard(1733) attributes(2) 4 }

-- ats-hash-index attribute
id-aa-ATSHashIndex OBJECT IDENTIFIER ::= { itu-t(0)
  identified-organization(4) etsi(0)
  electronic-signature-standard(1733) attributes(2) 5 }
```

```
ATSHashIndex ::= SEQUENCE {
        hashIndAlgorithm AlgorithmIdentifier DEFAULT {algorithm id-sha256},
        certificatesHashIndex    SEQUENCE OF OCTET STRING,
        crlsHashIndex            SEQUENCE OF OCTET STRING,
        unsignedAttrsHashIndex   SEQUENCE OF OCTET STRING
}

-- Long-Term Validation attribute

id-aa-ets-longTermValidation OBJECT IDENTIFIER ::= { itu-t(0) identified-organization(4) etsi(0)
       electronic-signature-standard (1733) attributes(2) 2 }

LongTermValidation ::= SEQUENCE {
    poeDate    GeneralizedTime,
    poeValue   CHOICE {
        timeStamp         [0] EXPLICIT TimeStampToken,
        evidenceRecord    [1] EXPLICIT EvidenceRecord
    } OPTIONAL,
    extraCertificates    [0] IMPLICIT CertificateSet OPTIONAL,
    extraRevocation      [1] IMPLICIT RevocationInfoChoices OPTIONAL
}



-- Attribute-certificate-references attribute

id-aa-ets-attrCertificateRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 44}

AttributeCertificateRefs ::=  SEQUENCE OF OtherCertID

-- Attribute-revocation-references attribute

id-aa-ets-attrRevocationRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 45}

AttributeRevocationRefs ::=  SEQUENCE OF CrlOcspRef


END
```

E.2 Signature Format Definitions Using X.680 ASN.1 Syntax

NOTE: The ASN.1 module defined in clause A.1 has precedence over that defined in clause A.2 using syntax defined in Recommendation ITU-T X.680 [11] in the case of any conflict.

```
ETS-ElectronicSignatureFormats-ExplicitSyntax97 { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-mod(0) eSignature-explicit97(29)}

DEFINITIONS EXPLICIT TAGS ::=
BEGIN
-- EXPORTS All -

IMPORTS

-- Cryptographic Message Syntax (CMS): RFC 5652

    ContentInfo, ContentType, id-data, id-signedData, SignedData,
    EncapsulatedContentInfo, SignerInfo,
    id-contentType, id-messageDigest, MessageDigest, id-signingTime, SigningTime,
    id-countersignature, Countersignature
   FROM CryptographicMessageSyntax2004
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) cms-2004(24) }


-- ESS Defined attributes: ESS Update
-- RFC 5035 (Adding CertID Algorithm Agility)

   id-aa-signingCertificate, SigningCertificate, IssuerSerial,
    id-aa-contentReference, ContentReference, id-aa-contentIdentifier, ContentIdentifier,
    id-aa-signingCertificateV2
      FROM ExtendedSecurityServices-2006
        { iso(1) member-body(2) us(840) rsadsi(113549)
          pkcs(1) pkcs-9(9) smime(16) modules(0) id-mod-ess-2006(30) }



-- Internet X.509 Public Key Infrastructure - Certificate and CRL Profile: RFC 5280

    Certificate, AlgorithmIdentifier, CertificateList, Name,
    Attribute
  FROM PKIX1Explicit88
    {iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-explicit(18)}

   GeneralNames, GeneralName, PolicyInformation
     FROM PKIX1Implicit88 {iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-implicit(19)}


-- Internet Attribute Certificate Profile for Authorization: RFC 5755

AttributeCertificate
FROM PKIXAttributeCertificate
     {iso(1) identified-organization(3) dod(6)  internet(1)
     security(5) mechanisms(5) pkix(7) id-mod(0)  id-mod-attribute-cert(12)}

-- OCSP RFC 6960

BasicOCSPResponse, ResponderID
    FROM OCSP {iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-ocsp(14)}


-- RFC 3161 Internet X.509 Public Key Infrastructure
-- Time-Stamp Protocol
TimeStampToken
    FROM PKIXTSP {iso(1) identified-organization(3) dod(6) internet(1)
   security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-tsp(13)}

-- Evidence Record Syntax RFC 4998
EvidenceRecord
    FROM ERS
    {iso(1) identified-organization(3) dod(6) internet(1) security(5)
    mechanisms(5) ltans(11) id-mod(0) id-mod-ers88(2) id-mod-ers88-v1(1)}

-- X.520

    DirectoryString {}
    FROM SelectedAttributeTypes
       {joint-iso-itu-t ds(5) module(1) selectedAttributeTypes(5) 4}

;



-- Definitions of Object Identifier arcs used in the present document
-- ==================================================================

-- OID used referencing electronic signature mechanisms based on the present document
-- for use with the IDUP API (see Annex D)

id-etsi-es-IDUP-Mechanism-v1 OBJECT IDENTIFIER ::=
    { itu-t(0) identified-organization(4) etsi(0)
       electronic-signature-standard (1733) part1 (1) idupMechanism (4) etsiESv1(1) }

id-etsi-es-attributes OBJECT IDENTIFIER ::=
    { itu-t(0) identified-organization(4) etsi(0)
       electronic-signature-standard (1733) attributes(2) }


-- Basic ES Attributes Defined in the present document
-- ===================================================


-- CMS Attributes defined in the present document
-- OtherSigningCertificate - deprecated

id-aa-ets-otherSigCert OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-aa(2) 19 }

OtherSigningCertificate ::=  SEQUENCE {
    certs         SEQUENCE OF OtherCertID,
    policies      SEQUENCE OF PolicyInformation OPTIONAL
                  -- NOT USED IN THE PRESENT DOCUMENT
    }

OtherCertID ::= SEQUENCE {
     otherCertHash            OtherHash,
     issuerSerial             IssuerSerial OPTIONAL }

OtherHash ::= CHOICE {
    sha1Hash OtherHashValue,  -- This contains a SHA-1 hash
    otherHash OtherHashAlgAndValue}

-- Policy ES Attributes Defined in the present document
-- ===================================================

-- Mandatory Basic Electronic Signature Attributes, plus in addition.
-- Signature Policy Identifier

id-aa-ets-sigPolicyId OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-aa(2) 15 }

SignaturePolicyIdentifier  ::= CHOICE {
         signaturePolicyId          SignaturePolicyId,
         signaturePolicyImplied     SignaturePolicyImplied
                                    -- not used in this version

}

 SignaturePolicyId ::= SEQUENCE {
         sigPolicyId           SigPolicyId,
         sigPolicyHash         SigPolicyHash,

        sigPolicyQualifiers   SEQUENCE SIZE (1..MAX) OF
                                 SigPolicyQualifierInfo OPTIONAL
}


SignaturePolicyImplied ::= NULL


SigPolicyId ::= OBJECT IDENTIFIER

SigPolicyHash ::= OtherHashAlgAndValue

   OtherHashAlgAndValue ::= SEQUENCE {
      hashAlgorithm   AlgorithmIdentifier,
      hashValue       OtherHashValue }

   OtherHashValue ::= OCTET STRING


SigPolicyQualifierInfo ::= SEQUENCE {
        sigPolicyQualifierId       SIG-POLICY-QUALIFIER.&id
                ({SupportedSigPolicyQualifiers}),
        qualifier               SIG-POLICY-QUALIFIER.&Qualifier
                                ({SupportedSigPolicyQualifiers}
                                    {@sigPolicyQualifierId})OPTIONAL }

SupportedSigPolicyQualifiers SIG-POLICY-QUALIFIER ::= { noticeToUser |
                                     pointerToSigPolSpec }

SIG-POLICY-QUALIFIER ::= CLASS {
        &id             OBJECT IDENTIFIER UNIQUE,
        &Qualifier      OPTIONAL }
WITH SYNTAX {
        SIG-POLICY-QUALIFIER-ID     &id
        [SIG-QUALIFIER-TYPE &Qualifier] }

noticeToUser SIG-POLICY-QUALIFIER ::= {
      SIG-POLICY-QUALIFIER-ID id-spq-ets-unotice SIG-QUALIFIER-TYPE SPUserNotice }

pointerToSigPolSpec SIG-POLICY-QUALIFIER ::= {
      SIG-POLICY-QUALIFIER-ID id-spq-ets-uri SIG-QUALIFIER-TYPE SPuri }


    id-spq-ets-uri OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-spq(5) 1 }

SPuri ::= IA5String

    id-spq-ets-unotice OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-spq(5) 2 }

   SPUserNotice ::= SEQUENCE {
        noticeRef        NoticeReference OPTIONAL,
        explicitText     DisplayText OPTIONAL}

   NoticeReference ::= SEQUENCE {
        organization     DisplayText,
        noticeNumbers    SEQUENCE OF INTEGER }

   DisplayText ::= CHOICE {
        visibleString    VisibleString  (SIZE (1..200)),
        bmpString        BMPString      (SIZE (1..200)),
        utf8String       UTF8String     (SIZE (1..200)) }

SPDocSpecification ::= CHOICE {
        oid     OBJECT IDENTIFIER,
        uri     IA5String }

    id-spq-ets-docspec OBJECT IDENTIFIER ::=  xxxx



-- Optional Electronic Signature Attributes

-- Signature-policy-store attribute
id-aa-ets-sigPolicyStore OBJECT IDENTIFIER ::= xxxx

SignaturePolicyStore ::=SEQUENCE {
        spDocSpec     SPDocSpecification ,
        spDocument      SignaturePolicyDocument }

SignaturePolicyDocument ::= CHOICE {
        sigPolicyEncoded      OCTET STRING,
        sigPolicyLocalURI     IA5String }


-- Commitment Type

id-aa-ets-commitmentType OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 16}

CommitmentTypeIndication ::= SEQUENCE {
  commitmentTypeId CommitmentTypeIdentifier,
  commitmentTypeQualifier SEQUENCE SIZE (1..MAX) OF CommitmentTypeQualifier OPTIONAL}

CommitmentTypeIdentifier ::= OBJECT IDENTIFIER

CommitmentTypeQualifier ::= SEQUENCE {
        commitmentQualifierId       COMMITMENT-QUALIFIER.&id,
        qualifier               COMMITMENT-QUALIFIER.&Qualifier OPTIONAL }

COMMITMENT-QUALIFIER ::= CLASS {
        &id             OBJECT IDENTIFIER UNIQUE,
        &Qualifier      OPTIONAL }
WITH SYNTAX {
         COMMITMENT-QUALIFIER-ID     &id
        [COMMITMENT-TYPE &Qualifier] }


    id-cti-ets-proofOfOrigin OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 1}
    id-cti-ets-proofOfReceipt OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 2}

    id-cti-ets-proofOfDelivery OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 3}

    id-cti-ets-proofOfSender OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 4}

    id-cti-ets-proofOfApproval OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 5}

    id-cti-ets-proofOfCreation OBJECT IDENTIFIER ::= { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) cti(6) 6}


-- Signer Location

id-aa-ets-signerLocation OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 17}

SignerLocation ::= SEQUENCE {
    -- at least one of the following shall be present
            countryName [0] DirectoryString OPTIONAL,
                -- As used to name a Country in X.500
        localityName [1] DirectoryString OPTIONAL,
                -- As used to name a locality in X.500
            postalAdddress [2] PostalAddress OPTIONAL }

    PostalAddress ::= SEQUENCE SIZE(1..6) OF DirectoryString{maxSize}
    -- maxSize parametrization as specified in X.683


-- Signer Attributes

id-aa-ets-signerAttr OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 18}

SignerAttribute ::= SEQUENCE OF CHOICE {
            claimedAttributes    [0] ClaimedAttributes,
            certifiedAttributes [1] CertifiedAttributes }

ClaimedAttributes ::= SEQUENCE OF Attribute

CertifiedAttributes ::= AttributeCertificate  -- as defined in RFC 5755: see clause 4.1.


-- Signer-attributes-v2 attribute

id-aa-ets-signerAttrV2 OBJECT IDENTIFIER ::= xxxx

SignerAttributeV2 ::= SEQUENCE {
            claimedAttributes       [0] ClaimedAttributes OPTIONAL,
            certifiedAttributesV2   [1] CertifiedAttributesV2 OPTIONAL,
            signedAssertions        [2] SignedAssertions OPTIONAL}

ClaimedAttributes ::= SEQUENCE OF Attribute

CertifiedAttributesV2 ::= SEQUENCE OF CHOICE {
            attributeCertificate        [0] AttributeCertificate,
                                            -- as defined in RFC 3281: see clause 4.1.
            otherAttributeCertificate   [1] OtherAttributeCertificate }

OtherAttributeCertificate ::= SEQUENCE {
        otherAttributeCertID        OTHER-ATTRIBUTE-CERT.&id,
        otherAttributeCert          OTHER-ATTRIBUTE-CERT.&OtherAttributeCert OPTIONAL }

OTHER-ATTRIBUTE-CERT ::= CLASS {
        &id                     OBJECT IDENTIFIER UNIQUE,
        &OtherAttributeCert     OPTIONAL }
WITH SYNTAX {
         OTHER-ATTRIBUTE-CERT-ID    &id
        [OTHER-ATTRIBUTE-CERT-TYPE  &OtherAttributeCert] }

SignedAssertions ::= SEQUENCE OF SignedAssertion

SignedAssertion ::= SEQUENCE {
        signedAssertionID       SIGNED-ASSERTION.&id,
        signedAssertion         SIGNED-ASSERTION.&Assertion OPTIONAL }

SIGNED-ASSERTION::= CLASS {
        &id             OBJECT IDENTIFIER UNIQUE,
        &Assertion      OPTIONAL }
WITH SYNTAX {
         SIGNED-ASSERTION-ID    &id
        [SIGNED-ASSERTION-TYPE  &Assertion] }

-- claimed-SAML-assertion

id-aa-ets-claimedSAML OBJECT IDENTIFIER ::= xxxx

ClaimedSAMLAssertion ::= OCTET STRING.

-- Content Timestamp

id-aa-ets-contentTimestamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 20}

ContentTimestamp::= TimeStampToken

-- Mime type

id-aa-ets-mimeType OBJECT IDENTIFIER ::= { itu-t(0) identified-organization(4) etsi(0)
       electronic-signature-standard (1733) attributes(2) 1}

MimeType::= UTF8String

-- Signature Timestamp

id-aa-signatureTimeStampToken OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 14}

SignatureTimeStampToken ::= TimeStampToken

-- Complete Certificate Refs.

id-aa-ets-certificateRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 21}

CompleteCertificateRefs ::=  SEQUENCE OF OtherCertID


-- Complete Revocation Refs

id-aa-ets-revocationRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 22}

CompleteRevocationRefs ::=  SEQUENCE OF CrlOcspRef

CrlOcspRef ::= SEQUENCE {
    crlids          [0] CRLListID   OPTIONAL,
    ocspids         [1] OcspListID  OPTIONAL,
    otherRev        [2] OtherRevRefs OPTIONAL
}

CRLListID ::=  SEQUENCE {
    crls        SEQUENCE OF CrlValidatedID}

CrlValidatedID ::=  SEQUENCE {
     crlHash                    OtherHash,
     crlIdentifier              CrlIdentifier OPTIONAL}

CrlIdentifier ::= SEQUENCE {
    crlissuer                  Name,
    crlIssuedTime              UTCTime,
    crlNumber                  INTEGER OPTIONAL
}

OcspListID ::=  SEQUENCE {
    ocspResponses        SEQUENCE OF OcspResponsesID}
```

```
OcspResponsesID ::=  SEQUENCE {
    ocspIdentifier              OcspIdentifier,
    ocspRefHash                 OtherHash    OPTIONAL
}

OcspIdentifier ::= SEQUENCE {
    ocspResponderID     ResponderID,    -- As in OCSP response data
    producedAt          GeneralizedTime -- As in OCSP response data
}

OtherRevRefs ::= SEQUENCE {
    otherRevRefType OTHER-REVOCATION-REF.&id,
    otherRevRefs    SEQUENCE OF OTHER-REVOCATION-REF.&Type
 }

OTHER-REVOCATION-REF ::= CLASS {
        &Type,
        &id OBJECT IDENTIFIER UNIQUE }
    WITH SYNTAX {
        WITH SYNTAX &Type ID &id }


-- Certificate Values

id-aa-ets-certValues OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 23}

CertificateValues ::=  SEQUENCE OF Certificate

-- Certificate Revocation Values

id-aa-ets-revocationValues OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 24}

RevocationValues ::=  SEQUENCE {
    crlVals          [0] SEQUENCE OF CertificateList OPTIONAL,
    ocspVals             [1] SEQUENCE OF BasicOCSPResponse OPTIONAL,
    otherRevVals         [2] OtherRevVals OPTIONAL}

OtherRevVals ::= SEQUENCE {
    otherRevValType OTHER-REVOCATION-VAL.&id,
    otherRevVals    SEQUENCE OF OTHER-REVOCATION-REF.&Type
 }

OTHER-REVOCATION-VAL ::= CLASS {
        &Type,
        &id OBJECT IDENTIFIER UNIQUE }
    WITH SYNTAX {
        WITH SYNTAX &Type ID &id }


-- CAdES-C Timestamp

id-aa-ets-escTimeStamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 25}

ESCTimeStampToken ::= TimeStampToken


-- Time-Stamped Certificates and CRLs

id-aa-ets-certCRLTimestamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 26}

TimestampedCertsCRLs ::= TimeStampToken


-- Archive Timestamp

id-aa-ets-archiveTimestampV2 OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 48}

ArchiveTimeStampToken ::= TimeStampToken

-- archive-time-stamp-v3 attribute
id-aa-ets-archiveTimestampV3 OBJECT IDENTIFIER ::= { itu-t(0)
  identified-organization(4) etsi(0)
  electronic-signature-standard(1733) attributes(2) 4 }
```

```
-- ats-hash-index attribute
id-aa-ATSHashIndex OBJECT IDENTIFIER ::= { itu-t(0)
  identified-organization(4) etsi(0)
  electronic-signature-standard(1733) attributes(2) 5 }

ATSHashIndex ::= SEQUENCE {
        hashIndAlgorithm AlgorithmIdentifier DEFAULT {algorithm id-sha256},
        certificatesHashIndex    SEQUENCE OF OCTET STRING,
        crlsHashIndex            SEQUENCE OF OCTET STRING,
        unsignedAttrsHashIndex   SEQUENCE OF OCTET STRING
}

-- Long-Term Validation attribute

id-aa-ets-longTermValidation OBJECT IDENTIFIER ::= { itu-t(0) identified-organization(4) etsi(0)
       electronic-signature-standard (1733) attributes(2) 2 }

LongTermValidation ::= SEQUENCE {
    poeDate    GeneralizedTime,
    poeValue   CHOICE {
        timeStamp        [0] EXPLICIT TimeStampToken,
        evidenceRecord   [1] EXPLICIT EvidenceRecord
    } OPTIONAL,
    extraCertificates    [0] IMPLICIT CertificateSet OPTIONAL,
    extraRevocation      [1] IMPLICIT RevocationInfoChoices OPTIONAL
}-- Attribute certificate references

id-aa-ets-attrCertificateRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 44}

AttributeCertificateRefs ::=  SEQUENCE OF OtherCertID

-- Attribute revocation references

id-aa-ets-attrRevocationRefs OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 45}

AttributeRevocationRefs ::=  SEQUENCE OF CrlOcspRef


END
```

Annex F (informative): Example Structured Contents and MIME

F.1 Use of MIME to Encode Data

The signed content may be structured using MIME (Multipurpose Internet Mail Extensions - RFC 2045 [1]). Whilst the MIME structure was initially developed for Internet email, it has a number of features that make it useful to provide a common structure for encoding a range of electronic documents and other multi-media data (e.g. photographs, video). These features include:

- providing a means of signalling the type of "object" being carried (e.g. text, image, ZIP file, application data);
- providing a means of associating a file name with an object;
- associating several independent objects (e.g. a document and image) to form a multi-part object;
- handling data encoded in text or binary and, if necessary, re-encoding the binary as text.

When encoding a single object, MIME consists of:

- header information, followed by;
- encoded content.

This structure can be extended to support multi-part content.

F.1.1 Header Information

A MIME header includes:

MIME Version information:

```
e.g.: MIME-Version: 1.0
```

Content type information, which includes information describing the content sufficient for it to be presented to a user or application process, as required. This includes information on the "media type" (e.g. text, image, audio) or whether the data is for passing to a particular type of application. In the case of text, the content type includes information on the character set used.

```
e.g. Content-Type: text/plain; charset="us-ascii"
```

Content-encoding information, which defines how the content is encoded (see below about encoding supported by MIME).

Other information about the content, such as a description or an associated file name.

An example MIME header for text object is:

```
Mime-Version: 1.0
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: quoted-printable
```

An example MIME header for a binary file containing a PDF document is:

```
Content-Type: application/pdf
Content-Transfer-Encoding: base64
Content-Description: JCFV201.pdf
Content-Disposition: filename="JCFV201.pdf"
```

F.1.2 Content Encoding

MIME supports a range of mechanisms for encoding both text and binary data.

Text data can be carried transparently as lines of text data encoded in 7- or 8-bit ASCII characters. MIME also includes a "quoted-printable" encoding that converts characters other than the basic ASCII into an ASCII sequence.

Binary can either be carried:

- transparently as 8-bit octets; or
- converted to a basic set of characters using a system called Base64.

NOTE: As there are some mail relays that can only handle 7-bit ASCII, Base64 encoding is usually used on the Internet.

F.1.3 Multi-Part Content

Several objects (e.g. text and a file attachment) can be associated together using a special "multi-part" content type. This is indicated by the content type "multipart" with an indication of the string to be used indicating a separation between each part.

In addition to a header for the overall multipart content, each part includes its own header information indicating the inner content type and encoding.

An example of a multipart content is:

```
Mime-Version: 1.0
Content-Type: multipart/mixed; boundary="----=_NextPart_000_01BC4599.98004A80"
Content-Transfer-Encoding: 7bit

------=_NextPart_000_01BC4599.98004A80
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: 7bit

Per your request, I've attached our proposal for the Java Card Version
2.0 API and the Java Card FAQ.

------=_NextPart_000_01BC4599.98004A80
Content-Type: application/pdf; name="JCFV201.pdf"
Content-Transfer-Encoding: base64
Content-Description: JCFV201.pdf
Content-Disposition: attachment; filename="JCFV201.pdf"

0M8R4KGxGuEAAAAAAAAAAAAAAAAAAAAAPgADAP7/CQAGAAAAAAAAAAAAAAACAAAAAgAAAAAAAAAA
EAAAtAAAAAEAAAD+////AAAAAAMAAAAGAAAA////////////////////////////////////////
AANhAAQAYg==

------=_NextPart_000_01BC4599.98004A80--
```

Multipart content can be nested. So a set of associated objects (e.g. HTML text and images) can be handled as a single attachment to another object (e.g. text).

The Content-Type from each part of the MIME message indicates the type of content.

F.2 S/MIME

The specific use of MIME to carry CMS (extended as defined in the present document) secured data is called S/MIME (see RFC 3851 [i.21]).

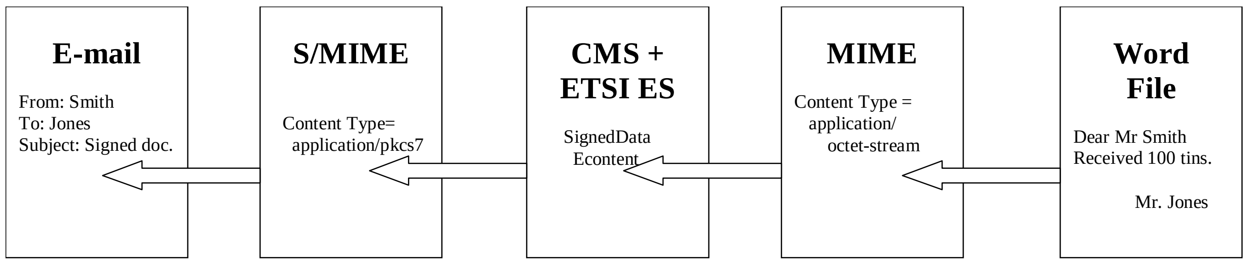


Figure F.1: Illustration of relation of using S/MIME

S/MIME carries electronic signatures as either:

- an "application/pkcs7-mime" object with the CMS carried as binary attachment (PKCS7 is the name of the early version of CMS).
  - The signed data may be included in the SignedData, which itself may be included in a single S/MIME object. See RFC 3851 [i.21], clause 3.4.2: "Signing Using application/pkcs7-mime with SignedData" and figure F.2.

or

- a "multipart/signed" object with the signed data and the signature encoded as separate MIME objects.
  - The signed data is not included in the SignedData, and the CMS structure only includes the signature. See RFC 3851 [i.21], clause 3.4.3: "Signing Using the multipart/signed Format" and figure F.3.

```
+-------------++----------++-------------++------------+
 |             ||          ||             ||            |
 |   S/MIME    ||  CAdES   ||    MIME     ||  pdf file  |
 |             ||          ||             ||            |
 |Content-Type=||SignedData||Content-Type=||Dear MrSmith|
 |application/ || eContent ||application/ ||Received    |
 |pkcs7-mime   ||          ||pdf          ||  100 tins  |
 |             ||          ||             ||            |
 |smime-type=  ||    /|    ||      /|     ||  Mr.Jones  |
 |signed-data  ||   / -----+      / ------+             |
 |             ||   \ -----+      \ ------+             |
 |             ||    \|    ||      \|     |+------------+
 |             ||          |+-------------+
 |             |+----------+
 +-----------------+
```

Figure F.2: Signing Using application/pkcs7-mime

F.2.1 Using application/pkcs7-mime

This approach is similar to handling signed data as any other binary file attachment.

An example of signed data encoded using this approach is:

```
Content-Type: application/pkcs7-mime; smime-type=signed-data;
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

  567GhIGfHfYT6ghyHhHUujpfyF4f8HHGTrfvhJhjH776tbB9HG4VQbnj7
  77n8HHGT9HG4VQpfyF467GhIGfHfYT6rfvbnj756tbBghyHhHUujhJhjH
  HUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H7n8HHGghyHh
  6YT64V0GhIGfHfQbnj75
```

F.2.2 Using application/pkcs7-signature

CMS also supports an alternative structure where the signature and data being protected are separate MIME objects carried within a single message. In this case, the signed data is not included in the SignedData, and the CMS structure only includes the signature. See RFC 3851 [i.21], clause 3.4.3: "Signing Using the multipart/signed Format" and figure F.3 hereafter.

An example of signed data encoded using this approach is:

```
Content-Type: multipart/signed;
          protocol="application/pkcs7-signature";
          micalg=sha1; boundary=boundary42

       --boundary42
       Content-Type: text/plain

       This is a clear-signed message.

       --boundary42
       Content-Type: application/pkcs7-signature; name=smime.p7s
       Content-Transfer-Encoding: base64
       Content-Disposition: attachment; filename=smime.p7s

       ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
       4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
       n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
       7GhIGfHfYT64VQbnj756

       --boundary42--
```

With this second approach, the signed data passes through the CMS process and is carried as part of a multiple-parts signed MIME structure, as illustrated in figure F.3. The CMS structure just holds the electronic signature.

```
+---------------++----------++-------------++------------+
 |               ||          ||             ||            |
 |     MIME      ||  CAdES   ||    MIME     ||  pdf file  |
 |               ||          ||             ||            |
 |Content-Type=  ||SignedData||Content-Type=||Dear MrSmith|
 |multipart/     ||          ||application/ ||Received    |
 |signed         ||          ||pdf          ||  100 tins  |
 |        /|     ||          ||             ||            |
 |       / -------------------+        /|   ||  Mr.Jones  |
 |       \ -------------------+       / -----+            |
 |        \|     ||          ||       \ -----+            |
 |Content-Type=  ||          ||        \|   |+------------+
 |application/   ||          |+-------------+
 |pdf            ||          |
 |               ||          |
```

Figure F.3: Signing Using application/pkcs7-signature

This second approach (multipart/signed) has the advantage that the signed data can be decoded by any MIME-compatible system even if it does not recognize CMS-encoded electronic signatures.

F.3 Use of MIME in the signature

CAdES allows two ways to include the MIME type of the data to be signed, either in the `contentDescription` element of the `content-hints` attribute or in the `mime-type` attribute. The included MIME type allows to give information on how the driving application should decode or display the signed data. Thus these attributes allow allows to give the application useful information. In addition, including the MIME type into the signature can also help to prevent attacks based on the fact that a binary data file might change its meaning/use depending on the application used to process the data.

Which information of the MIME header is included as MIME type into the signature depends on the needs of the driving application. We will give two examples.

1) For most application, it will be sufficient to know the application corresponding to signed data, thus they will put only the application or only the Content-Type as MIME-type into the signature, for example

   - `application/pdf` or

   - `text/plain; charset="us-ascii"`

2) In the case that the driving application is interested in all the details of the MIME header, it might put the whole header as MIME-type into the signature, like for example:

```
Content-Type: application/pdf
Content-Transfer-Encoding: base64
Content-Description: JCFV201.pdf
Content-Disposition: filename="JCFV201.pdf"
```

Annex G (informative): Change History

date	Version	Information about changes
November 2013	V0.0.3	Restructuring of the document to align with EN 319 132, addition of attributes: `signer-attributesV2`, `signature-policy-storage`, `claimed-SAML-assertion;` definition of the `SPDocSpecification` qualifier and the introduction of different conformance levels. The Hash computation annex was removed to avoid inconsistencies. Several parts of TS 101 733 not related the core specification were removed. An example of using MIME type was added in annex F.

History

Document history		
V0.0.3	November 2013	Stable draft for public review