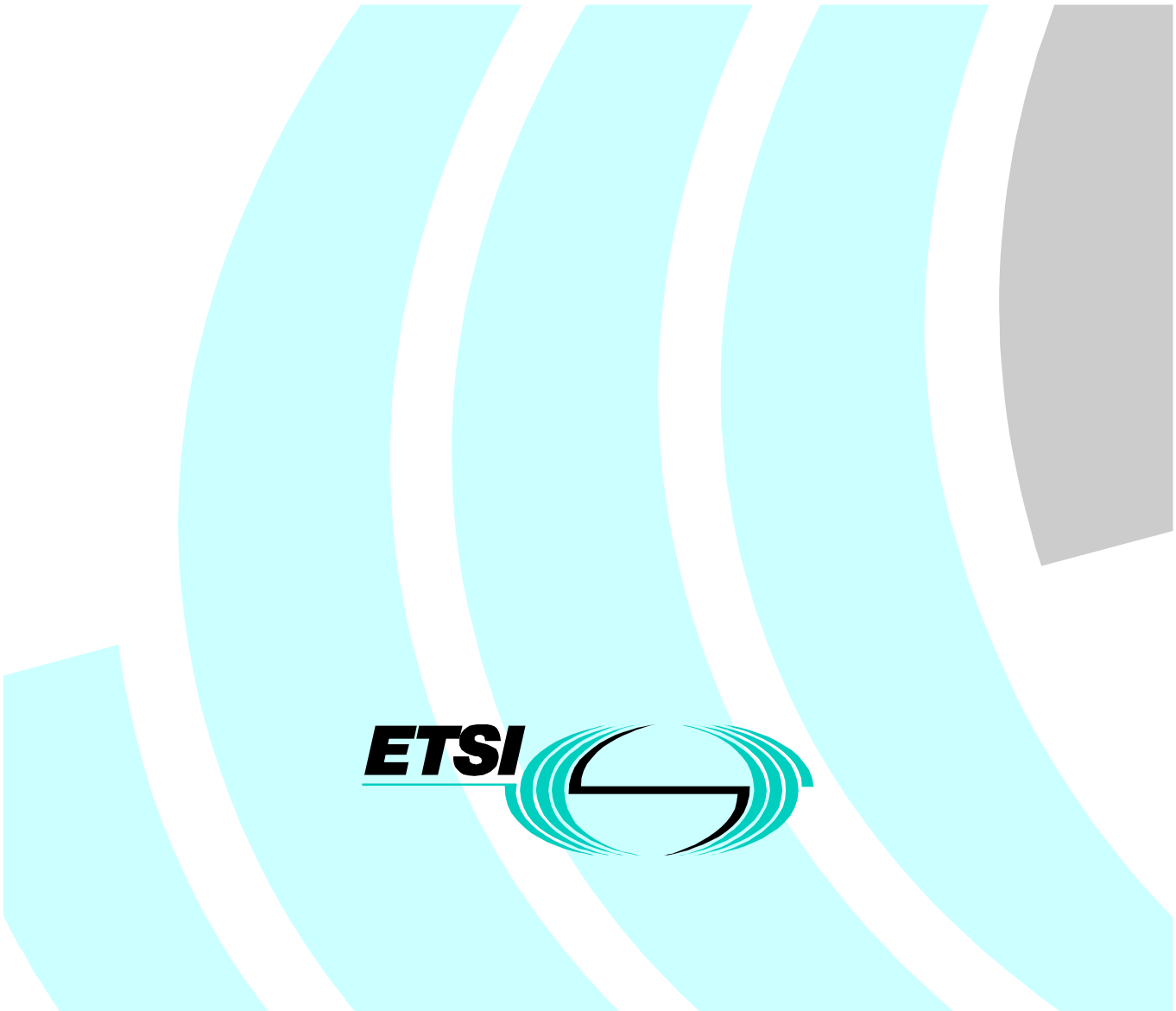
Draft ETSI ES 201 733 V1.1.2 (2000-01)

ETSI Standard

Electronic Signature Formats



Reference
DES/SEC-003007-1

Keywords
IP, electronic signature, security

ETSI

Postal address
F-06921 Sophia Antipolis Cedex - FRANCE

Office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - 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

Internet
secretariat@etsi.fr
Individual copies of this ETSI deliverable
can be downloaded from
http://www.etsi.org
If you find errors in the present document, send your
comment to: editor@etsi.fr

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://www.etsi.org/ipr).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Security (SEC), and is now submitted for the ETSI standards Membership Approval Procedure.

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 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.

An electronic signature produced in accordance with the present document provides evidence that can be processed to get confidence that some commitment has been explicitly endorsed under a Signature policy, at a given time, by a signer under an identifier, e.g. a name or a pseudonym, and optionally a role.

The European Directive on a community framework for Electronic Signatures 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". An electronic signature as used in the current document is a form of advanced electronic signature as defined in the Directive.

1 Scope

The scope of the present document includes:

- format of Electronic Signature tokens;
- format of sub-components of Electronic Signature tokens, e.g. Public Key Certificates, Attribute Certificates and Certificate Revocation Lists;
- format of Signature Policies.

In addition, the present document identifies other documents that define protocols for use of trusted third parties to support the operation of electronic signature creation and validation, as well as the management of certificates used to support electronic signatures.

Informative annexes, describe:

- an example structured content;
- the relationship between the present document and the directive on electronic signature and associated standardization initiatives;
- APIs to support the generation and the verification of electronic signatures;
- cryptographic algorithms that may be used;
- guidance on naming.

2 References

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

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.
- A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] ITU-T Recommendation X.509 (1997) | ISO/IEC 9594-8: "Information technology - Open Systems Interconnection - The Directory: authentication framework".

[2] CCITT Recommendation X.208 (1988): "Specification of Abstract Syntax Notation One (ASN.1)".

[3] ITU-T Recommendation X.680 (1994) | ISO/IEC 8824-1: "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

[4] ITU-T Recommendation X.690 (1997) | ISO/IEC 8825-1: "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)".

[5] ITU-T Recommendation F.1 (1998): "Operational provisions for the international public telegram service".

[6] IETF RFC 1777 (1995): "Lightweight Directory Access Protocol".

[7] IETF RFC 2459 (1999): "Internet X.509 Public Key Infrastructure Certificate and CRL Profile".

[8] IETF RFC 2560 (1999): "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP".

[9] IETF RFC 2630 (1999): "Cryptographic Message Syntax".

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

[11] IETF draft (draft-ietf-pkix-time-stamp-05.txt): "Internet X.509 Public Key Infrastructure - Time Stamp Protocol (TPS)".

NOTE 1: Not yet published.

[12] ISO 7498-2: "Information processing systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture".

[13] ISO/IEC 13888-1: "Information technology - Security techniques - Non-repudiation - Part 1: General".

[14] ITU-T Recommendation X.400: "Message handling system and service overview".

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

[16] ITU-T Recommendation X.501: "Information Technology - Open Systems Interconnection - The Directory: Models".

NOTE 2: Not yet published.

[17] ITU-T Recommendation X.520: "Information technology - Open Systems Interconnection - The Directory: Selected attribute types".

NOTE 3: Not yet published.

[18] IETF RFC 2559 (1999): "Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2".

[19] IETF RFC 2587 (1999): "Internet X.509 Public Key Infrastructure LDAPv2 Schema".

[20] IETF RFC 2510 (1999): "Internet X.509 Public Key Infrastructure Certificate Management Protocols".

[21] RFC 2450: "Proposed TLA and NLA Assignment Rules".

[22] RCF 2643: "Cabletron's SecureFast VLAN Operational Model, Version 1.8".

[23] RFC 1321: "The MD5 Message-Digest Algorithm".

[24] RFC 2246: "The TLS Protocol Version 1.0".

[25] RFC 2068: "Hypertext Transfer Protocol - HTTP/1.1".

[26] Void.

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

[28] RFC 1521: "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies".

[29] RFC 2078: "Generic Security Service Application Program Interface, Version 2".

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

[31] ANS X9.30-2(1997): "Public Key Cryptography for the Financial Services Industry - Part 2: The Secure Hash Algorithm (SHA-1)".

[32] RFC 2437: "PKCS #1: RSA Cryptography Specifications Version 2.0".

[33] ISO/IEC 10118-1 (1994): "Information technology - Security techniques - Hash-functions - Part 1: General".

[34] ISO/IEC 10118-2 (1994): "Information technology - Security techniques - Hash-functions - Part 2: Hash-functions using an n-bit block cipher algorithm".

[35] ISO/IEC 10118-3 (1997): "Information technology - Security techniques - Hash-functions - Part 3: Dedicated hash-functions".

[36] ISO/IEC FCD 10118-4: "Information technology - Security techniques - Hash-functions - Part 4: Hash-functions using modular arithmetic".

[37] IETF RFC 2511 (1999): "Internet X.509 Certificate Request Message Format".

[38] RFC 1320 (PS 1992): "The MD4 Message-Digest Algorithm".

[39] FIPS Publication 180-1 (1995): "Secure Hash Standard".

[40] ANS X9.31-2: "Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry - Part 2: Hash Algorithms".

NOTE 4: Not yet published.

[41] FIPS Publication 186 (1994): "Digital Signature Standard".

[42] IEEE P1363: "Standard Specifications for Public-Key Cryptography".

NOTE 5: Not yet published.

[43] ISO/IEC 9796 (1991): "Information technology - Security techniques - Digital signature scheme giving message recovery".

[44] ISO/IEC 9796-2 (1997): "Information technology - Security techniques - Digital signature schemes giving message recovery - Part 2: Mechanisms using a hash-function".

[45] ISO/IEC CD 9796-4: "Digital signature schemes giving message recovery - Part 4: Discrete logarithm based mechanisms".

NOTE 6: Not yet published.

[46] ISO/IEC FCD 14888-1: "Digital signatures with appendix - Part 1: General. Status".

NOTE 7: Not yet published.

[47] ISO/IEC FCD 14888-2: "Digital signatures with appendix - Part 2: Identity-based mechanisms".

NOTE 8: Not yet published.

[48] ISO/IEC FCD 14888-3: "Digital signatures with appendix - Part 3: Certificate-based mechanisms".

NOTE 9: Not yet published.

[49] ISO/IEC WD 15946-2: "Cryptographic techniques based on elliptic curves - Part 2: Digital signatures".

NOTE 10: Not yet published.

[50] ANS X9.31-1: "Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry - Part 1: The RSA Signature Algorithm".

NOTE 11: Not yet published.

[51] ANS X9.30-1 (1997): "Public Key Cryptography for the Financial Services Industry - Part 1: The Digital Signature Algorithm (DSA)".

[52] ANS X9.62 (draft): Public Key Cryptography for the Financial Services Industry - The Elliptic Curve Digital Signature Algorithm (ECDSA).

NOTE 12: Not yet published.

[53] IETF draft (draft-ietf-pkix-cmc-05.txt): "Certificate management Messages over CMS".

NOTE 13: Not yet published.

3 Definitions and abbreviations

3.1 Definitions

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

arbitrator: arbitrator entity may be used to arbitrate a dispute between a signer and verifier when there is a disagreement on the validity of a digital signature

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

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

Attribute Authority Revocation List (AARL): references to attribute certificates issued to AAs, that are no longer considered valid by the issuing authority

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

Certification Authority (CA): authority trusted by one or more users to create and assign certificates. Optionally the certification authority may create the users' keys (ITU-T Recommendation X.509 [1])

Certificate Revocation List (CRL): signed list indicating a set of certificates that are no longer considered valid by the certificate issuer [X.509 FPAM]

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 (ISO 7498-2 [12])

public key certificate: public keys of a user, together with some other information, rendered unforgeable by encipherment with the private key of the certification authority which issued it (ITU-T Recommendation X.509 [1])

signature policy: set of rules for the creation and validation of an electronic signature, under which the signature can be determined to be valid

signature policy issuer: entity that defines the technical and procedural requirements for electronic signature creation and validation, in order to meet a particular business need

signature validation policy: part of the signature policy which specifies the technical requirements on the signer in creating a signature and verifier when validating a signature

signer: signer is the entity that creates an electronic signature

TimeStamping Authority (TSA): TSA is a trusted third party that creates time stamp tokens in order to indicate that a datum existed at a particular point in time [11]

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

valid electronic signature: electronic signature which passes validation according to a signature validation policy

verifier: entity that verifies an evidence. (ISO/IEC 13888-1 [13]). Within the context of the present document this is an entity that validates an electronic signature

3.2 Abbreviations

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

AA	Attribute Authority
API	Application Program Interface
ARL	Authority Revocation List
ASN.1	Abstract Syntax Notation 1
CA	Certification Authority
CMS	Cryptographic Message Syntax
CRL	Certificate Revocation List
DER	Distinguished Encoding Rules (for ASN.1)
DSA	Digital Signature Algorithm (see annex E on crytpographic algorithms)
EDIFACT	Electronic Data Interchange for Administration, Commerce And Transport
ES	Electronic Signature
ES-A	ES with Archive Validation Data
ES-C	ES with Complete validation data
ESS	Enhanced Security Services (enhances CMS)
ES-T	ES with Timestamp
ES-X	ES with eXtended validation data
MIME	Multipurpose Internet Mail Extensions
OCSP	Online Certificate Status Provider
OID	Object Identifier
PIN	Personal Identification Number
PKI	Public Key Infrastructure
PKIX	Internet X.509 Public Key Infrastructure
SHA-1	Secure Hash Algorithm 1 (see annex E on crytpographic algorithms)
TSA	TimeStamping Authority
TSP	Trusted Service Provider
XML	eXtended Mark up Language

4 Overview

4.1 Aim

The aim of the present document is to define an electronic signature that remains 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 signature.

The present document specifies use of trusted service providers (e.g. TimeStamping Authorities), and the data that needs to be archived (e.g. cross certificates and revocation lists) to meet the requirements of long term electronic signatures. An electronic signature defined by the present document can 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 uses a signature policy, referenced by the signer, as the basis for establishing the validity of an electronic signature.

4.2 Basis of Present Document

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

The present document also uses timestamping services 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, as an option, uses additional timestamps to provide very long-term protection against key compromise or weakened algorithms.

The present document builds on existing standards that are widely adopted. This includes:

- RFC 2630 [9] "Crytographic Message Syntax (CMS)";
- ITU-T Recommendation X.509 [1] "Authentication framework";
- RFC 2459 [7] "Internet X.509 Public Key Infrastructure (PKIX) Certificate and CRL Profile";
- RFC (to be published) PKIX Timestamping protocol [11].

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

4.3 Major Parties

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

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

The **Signer** is the entity that creates the electronic signature. When the signer digitally signs over data using the prescribed format, this represents a commitment on behalf of the signing entity to the data being signed.

The **Verifier** is the entity that validates the electronic signature, it may be a single entity or multiple entities.

The **Trusted Service Providers** (TSPs) are one or more entities that help to build trust relationships between the signer and verifier. They support the signer and verifier by means of supporting services including: user certificates, cross-certificates, timestamping tokens, CRLs, ARLs, OCSP responses. The following TSPs are used to support the functions defined in the present document:

- Certification Authorities;

- Registration Authorities;
- Repository Authorities (e.g. a Directory);
- TimeStamping Authorities;
- Signature Policy Issuers.

Certification Authorities provide users with public key certificates.

Registration Authorities allows the registration of entities before a CA generates certificates.

Repository Authorities publish CRLs issued by CAs, signature policies issued by Signature Policy Issuers and optionally public key certificates.

TimeStamping Authorities attest that some data was formed before a given trusted time.

Signature Policy Issuers define the technical and procedural requirements for electronic signature creation and validation, in order to meet a particular business need.

In some cases the following additional TSPs are needed:

- Attribute Authorities.

Attributes Authorities provide users with attributes linked to public key certificates.

An **Arbitrator** is an entity that arbitrates disputes between a signer and a verifier.

4.4 Electronic Signatures and Validation Data

Validation of an electronic signature in accordance with the present document requires:

- The electronic signature; this includes:
  - the signature policy;
  - the signed user data;
  - the digital signature;
  - other signed attributes provided by the signer.
- Validation data which is the additional data needed to validate the electronic signature; this includes:
  - certificates;
  - revocation status information;
  - trusted time-stamps from Trusted Service Providers (TSPs).
- The **signature policy** specifies the technical requirements on signature creation and validation in order to meet a particular business need. A given legal/contractual context may recognize a particular signature policy as meeting its requirements. For example: a specific signature policy may be recognized by court of law as meeting the requirements of the European Directive for electronic commerce. A signature policy may be written using a formal notation like ASN.1 (see 11.1) or in an informal free text form provided the rules of the policy are clearly identified. However, for a given signature policy there shall be one definitive form which has a unique binary encoded value.

Signed user data is the user's data that is signed.

The **Digital Signature** is the digital signature applied over the following attributes provided by the signer:

- hash of the user data;
- signature Policy Identifier;
- other signed attributes.

The **other signed attributes** include any additional information which shall be signed to conform to the signature policy or the present document (e.g. signing time).

The **Validation Data** may be collected by the signer and/or the verifier and shall meet the requirements of the signature policy. Additional data includes CA certificates as well as revocation status information in the form of certificate revocation lists (CRLs) or certificate status information provided by an on-line service. Additional data also includes timestamps and other time related data used to provide evidence of the timing of given events. It is required, as a minimum, that either the signer or verifier obtains a timestamp over the signer's signature.

4.5 Forms of Validation Data

An electronic signature may exist in many forms including:

- the Electronic Signature (ES), which includes the digital signature and other basic information provided by the signer;
- the ES with Timestamp (ES-T), which adds a timestamp to the Electronic Signature, to take initial steps towards providing long term validity;
- the ES with Complete validation data (ES-C), which adds to the ES-T references to the complete set of data supporting the validity of the electronic signature (i.e. revocation status information).

The signer shall provide at least the ES form, but in some cases may decide to provide the ES-T form and in the extreme case could provide the ES-C form. If the signer does not provide ES-T, the verifier shall create the ES-T on first receipt of an electronic signature. The ES-T provides independent evidence of the existence of the signature at the time it was first verified which should be near the time it was created, and so protects against later repudiation of the existence of the signature. If the signer does not provide ES-C the verifier shall create the ES-C when the complete set of revocation and other validation data is available.

The ES satisfies the legal requirements for electronic signatures as defined in the European Directive on electronic signatures, see annex C for further discussion on relationship of the present document to the Directive. It provides basic authentication and integrity protection and can be created without accessing on-line (timestamping) services. However, without the addition of a timestamp the electronic signature does not protect against the threat that the signer later denies having created the electronic signature (i.e. does not provide non-repudiation of its existence).

The ES-T time-stamp should be created close to the time that ES was created to provide maximum protection against repudiation. At this time all the data needed to complete the validation may not be available but what information is readily available may be used to carry out some of the initial checks. For example, only part of the revocation information may be available for verification at that point in time.

Generally, the ES-C form cannot be created at the same time as the ES, as it is necessary to allow time for any revocation information to be captured. Also, if a certificate is found to be temporarily suspended, it will be necessary to wait until the end of the suspension period.

The signer should only create the ES-C in situations where it was prepared to wait for a sufficient length of time after creating the ES form before dispatching the ES-C. This, however, has the advantage that the verifier can be presented with the complete set of data supporting the validity of the ES.

Support for ES-C by the verifier is mandated (see clause 14 for specific conformance requirements).

An Electronic Signature (ES), with the additional validation data forming the ES-T and ES-C is illustrated in figure 1:

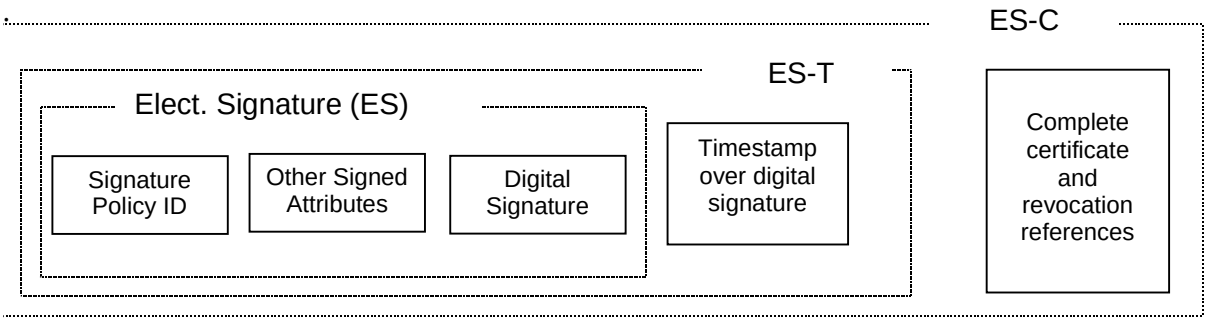


Figure 1: Illustration of an ES, ES-T and ES-C

4.6 Extended Forms of Validation Data

The complete validation data (ES-C) described above may be extended to form an ES with eXtended validation data (ES-X) to meet following additional requirements.

Firstly, when the verifier does not has access to:

- the signer's certificate,
- all the CA certificates that make up the full certification path,
- all the associated revocation status information, as referenced in the ES-C,

then the values of these certificates and revocation information may be added to the ES-C. This form of extended validation data is called a X-Long.

Secondly, if there is a risk that any CA keys used in the certificate chain may be compromised, then it is necessary to additionally timestamp the validation data by either:

- timestaming all the validation data as held with the ES(ES-C), this eXtended validation data is called a Type 1 X-Timestamp; or
- timestamping individual reference data as used for complete validation. This form of eXtended validation data is called a Type 2 X-Timestamp.

NOTE: The advantages/drawbacks for Type 1 and Type 2 X-Timestamp are discussed in subclause 5.4.6.

If all the above conditions occur then a combination of the two formats above may be used. This form of eXtended validation data is called a X-Long-Timestamped.

Support for the extended forms of validation data is optional.

An Electronic Signature (ES), with the additional validation data forming the ES-X long is illustrated in figure 2:

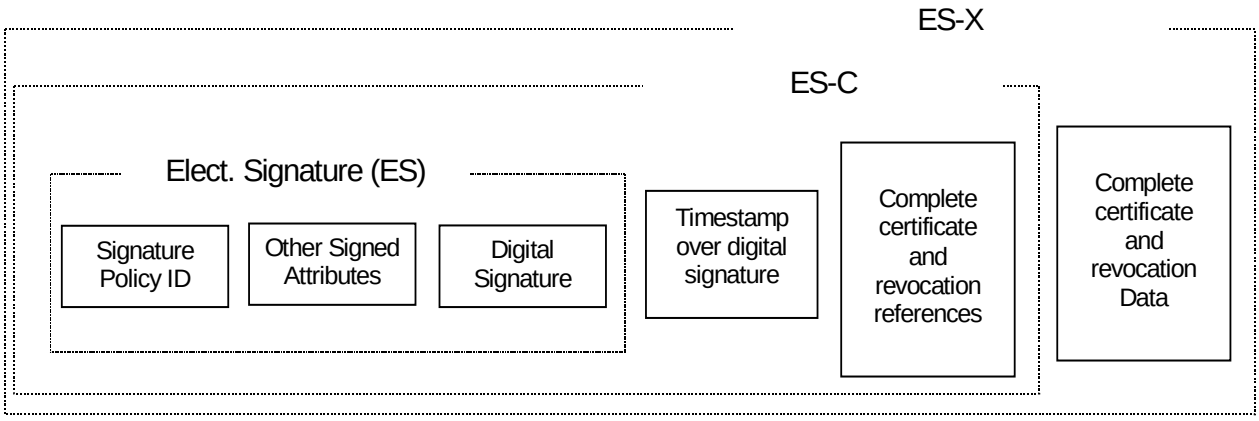


Figure 2: Illustration of an ES and ES-X long

An Electronic Signature (ES), with the additional validation data forming the eXtended Validation Data - Type 1 is illustrated in figure 3:

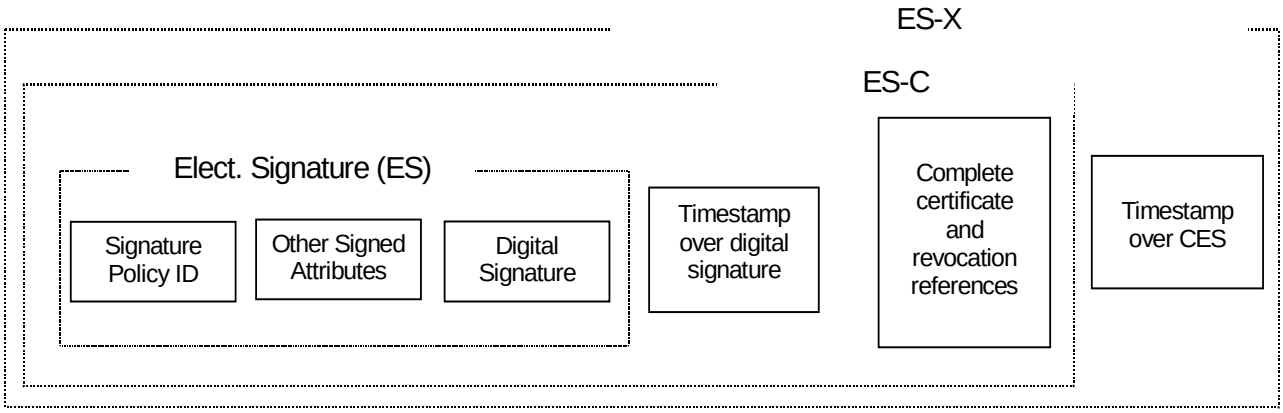


Figure 3: Illustration of ES with ES-X Type 1

An Electronic Signature (ES), with the additional validation data forming the eXtended Validation Data - Type 2 is illustrated in figure 4:

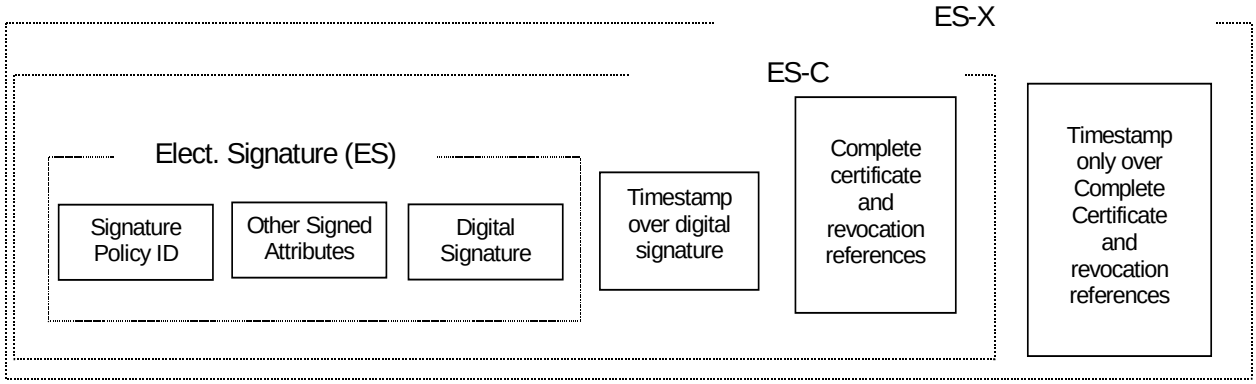


Figure 4: Illustration of ES with ES-X Type 2

4.7 Archive Validation Data

Before the algorithms, keys and other cryptographic data used at the time the ES-C was built become weak and the cryptographic functions become vulnerable, or the certificates supporting previous timestamps expires, the signed data, the ES-C and any additional information (ES-X) should be timestamped. If possible this should use stronger algorithms (or longer key lengths) than in the original timestamp. This additional data and timestamp is called Archive Validation Data. (ES-A). The Timestamping process may be repeated every time the protection used to timestamp a previous ES-A become weak. An ES-A may thus bear multiple embedded time stamps.

Support for ES-A is optional.

An example of an Electronic Signature (ES), with the additional validation data for the ES-C and ES-X forming the ES-A is illustrated in figure 5.

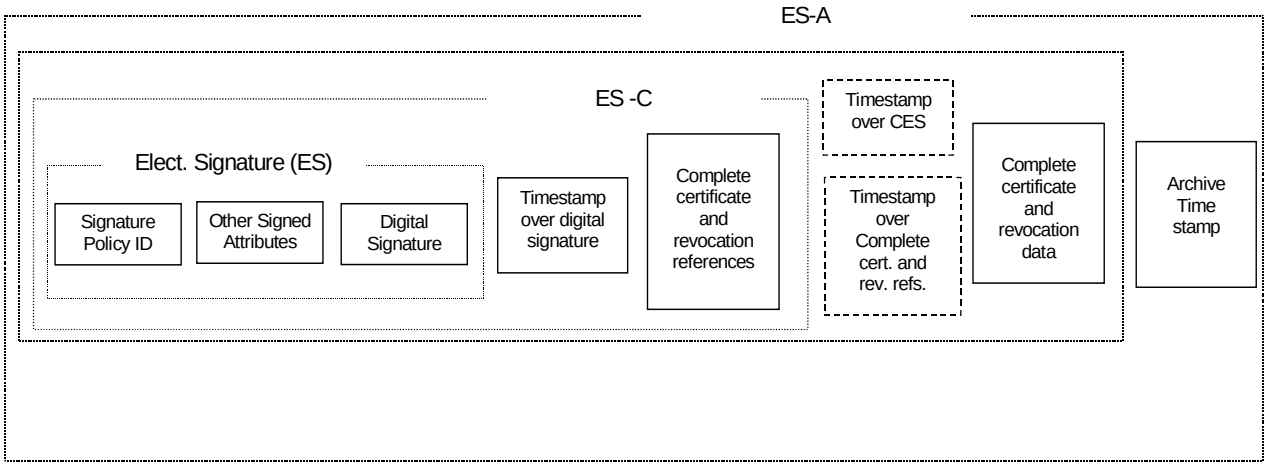


Figure 5: Illustration of ES-A

4.8 Arbitration

The ES-C may be used for arbitration should there be a dispute between the signer and verifier, provided that:

- the arbitrator knows where to retrieve the signer's certificate (if not already present), all the cross-certificates and the required CRLs and/or OCSPs responses referenced in the ES-C;
- none of the issuing key from the certificate chain have ever been compromised;
- the cryptography used at the time the ES-C was built has not been broken at the time the arbitration is performed.

When the first condition is not met, then the plaintiff shall provide an ES-X Long.

When it is known by some external means that the second condition is not met, then the plaintiff shall provide an ES-X Timestamped.

When the two previous conditions are not met, the plaintiff shall provide the two above information (i.e. an ES-X Timestamped and Long).

When the last condition is not met, the plaintiff shall provide an ES-A.

It should be noticed that a verifier may need to get **two time stamps** at two different instants of time: one soon after the generation of the ES and one soon after some grace period allowing any entity from the certification chain to declare a key compromise.

4.9 Validation Process

The **Validation Process** validates an electronic signature in accordance with the requirements of the signature policy. The output status of the validation process can be:

- valid;
- invalid;
- incomplete verification.

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

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

An **Incomplete Validation** response indicates that the format and digital signature verifications have not failed but there is insufficient information to determine if the electronic signature is valid under the signature policy. This can include situations where additional information, which does not effect the validity of the digital signature value, may be available but is invalid. In the case of Incomplete Validation, it may be possible to request that the electronic signature be checked again at a later date when additional validation information might become available. Also, in the case of incomplete validation, additional information may be made available to the application or user, thus allowing the application or user to decide what to do with partially correct electronic signatures.

The validation process may also output validation data:

- a signature timestamp;
- the complete validation data;
- the archive validation data.

4.10 Example Validation Sequence

As described earlier the signer or verifier may collect all the additional data that forms the Electronic Signature. Figure 6, and subsequent description, describes how the validation process may build up a complete electronic signature over time.

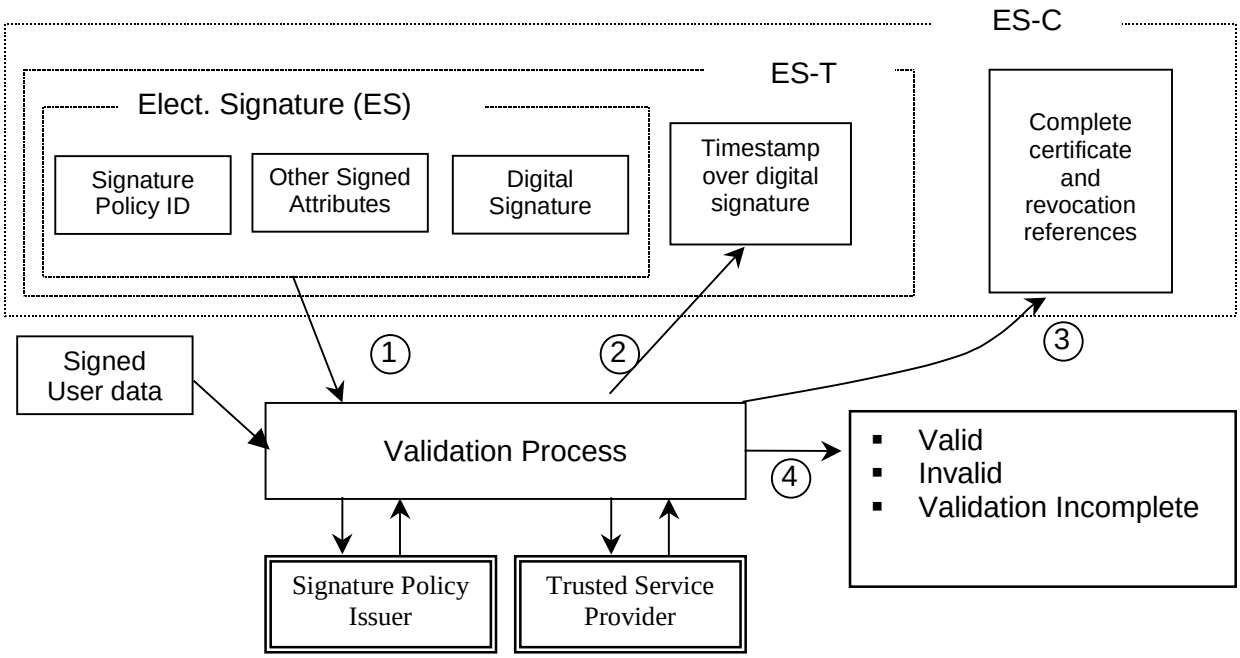


Figure 6: Illustration of an ES with Complete validation data

Soon after receiving the electronic signature (ES) from the signer (1), the digital signature value may be checked, the validation process shall at least add a time-stamp (2), unless the signer has provided one which is trusted by the verifier. The validation process may also validate the electronic signature, as required under the identified signature policy, using additional data (e.g. certificates, CRL, etc.) provided by trusted service providers. If the validation process is not complete then the output from this stage is the ES-T.

When all the additional data (e.g. the complete certificate and revocation information) necessary to validate the electronic signature first becomes available, then the validation process:

- obtains all the necessary additional certificate and revocation status information;
- completes all the validation checks on the ES, using the complete certificate and revocation information (if a timestamp is not already present, this may be added at the same stage combining ES-T and ES-C process);
- records the complete certificate and revocation references (3);
- indicates the validity status to the user (4).

At the same time as the validation process creates the ES-C, the validation process may provide and/or record the values of certificates and revocation status information used in ES-C, called the ES-X Long (5). This is illustrated in figure 7:

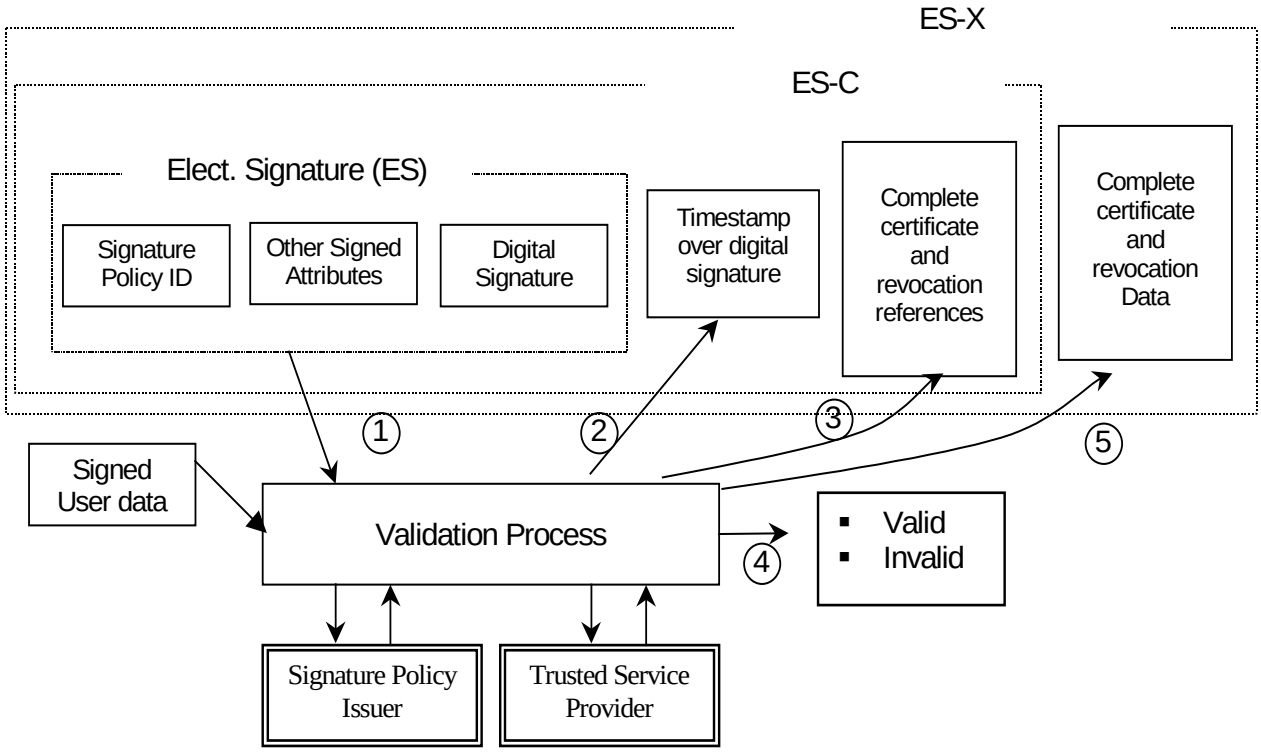


Figure 7: Illustration ES with eXtended Validation Data (Long)

When the validation process creates the ES-C it may also create extended forms of validation data. A first alternative is to timestamp all data forming the Type 1 X-Timestamp (6). This is illustrated in figure 8:

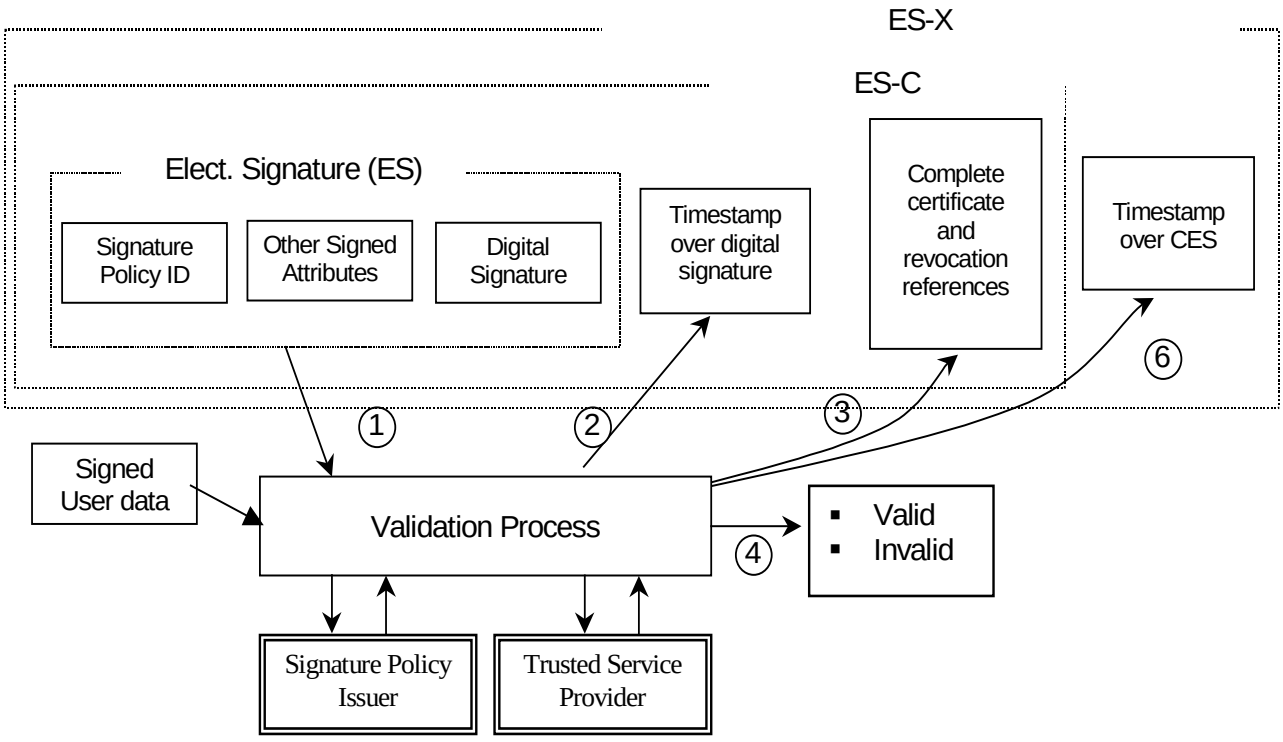


Figure 8: Illustration of ES with eXtended Validation Data - Type 1 X-Timestamp

Another alternative is to timestamp the certificate and revocation information references used to validate the electronic signature (but not the signature) (6'); this is called Type 2 X-Timestamped. This is illustrated in figure 9:

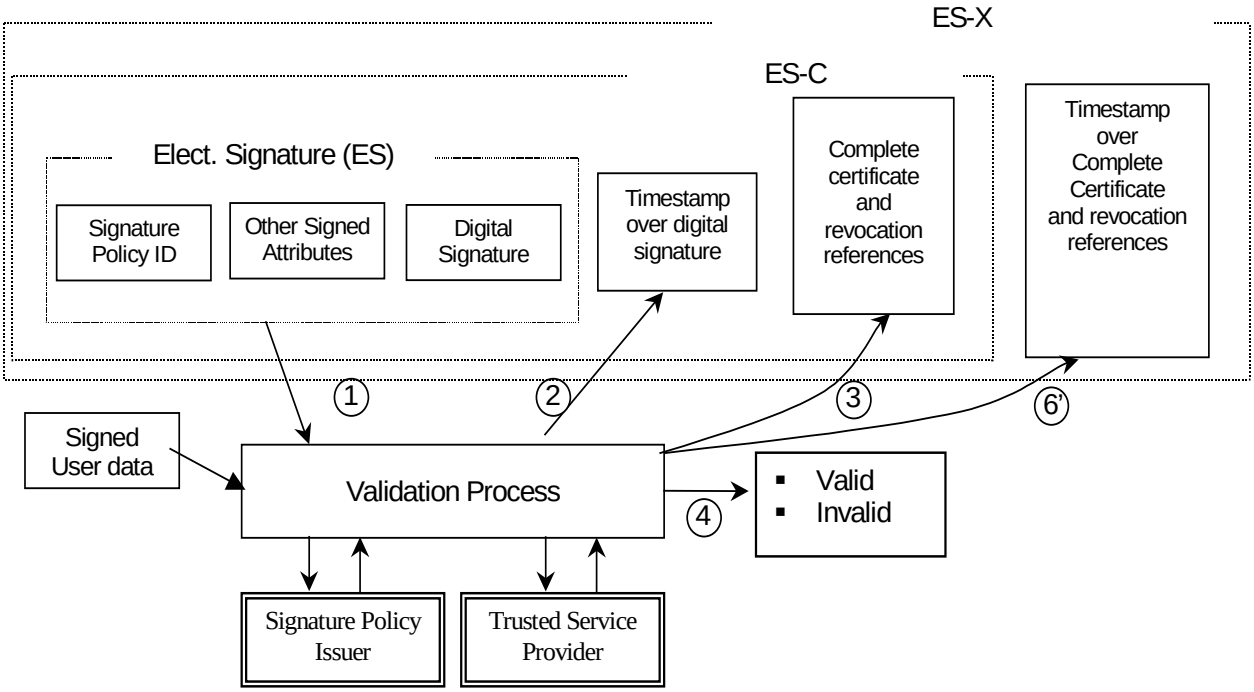


Figure 9: Illustration of ES with eXtended Validation Data - Type 2 X-Timestamp

Before the algorithms used in any of electronic signatures become or are likely, to be compromised or rendered vulnerable in the future, it is necessary to timestamp the entire electronic signature, including all the values of the validation and user data as an ES with Archive Validation Data (ES-A) (7). An ES-A is illustrated in figure 10:

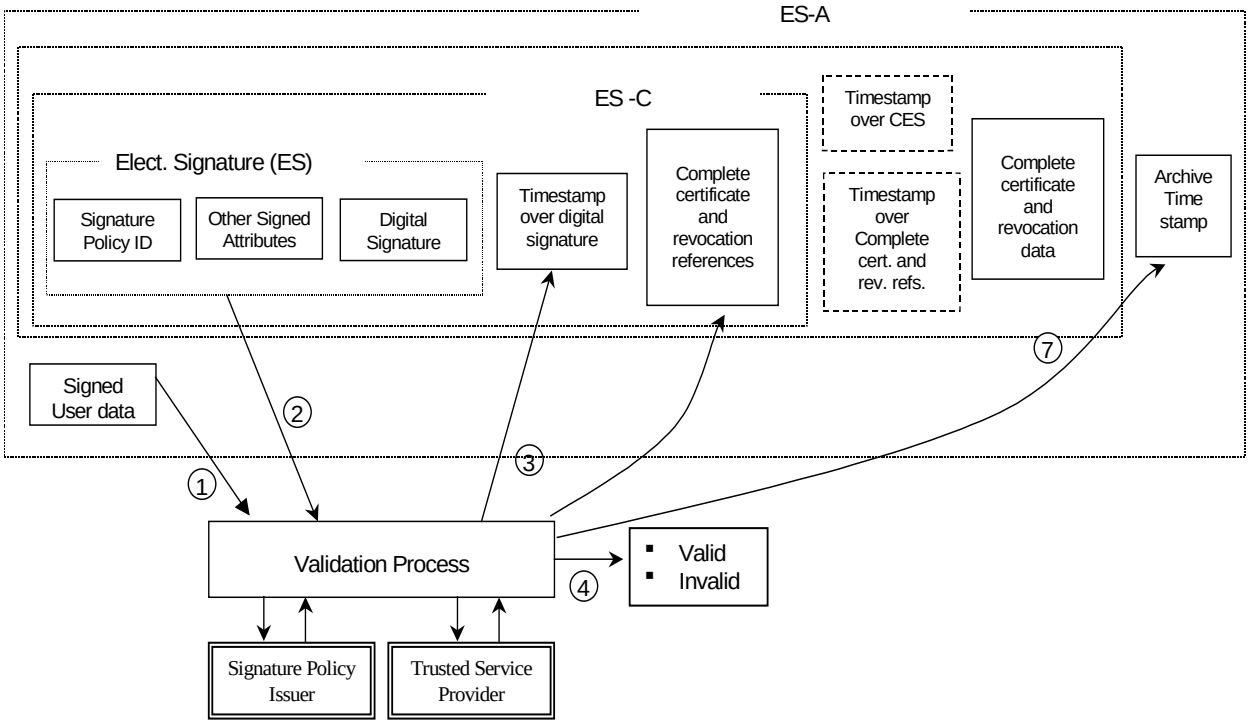


Figure 10: Illustration of an ES with Archive Validation Data

4.11 Additional optional features

The present document also defines additional optional features to:

- indicate a commitment type being made by the signer;
- indicate the role under which a signature was created;
- support multiple signatures.

5 General Description

This clause captures all the concepts that apply to the remaining of the document, in particular the rational for the clauses 8 and 9, that contain only the basic explanations of the ASN.1 components.

The specification below includes a description why the component is needed, with a brief description of the vulnerabilities and threats and the manner by which they are countered.

5.1 The Signature Policy

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.

A 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 needs to be in a computer processable form.

The signature policy thus includes the following:

- rules, which apply to functionality, covered by the present document (referred to as the Signature Validation Policy);
- rules which 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, which 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.

The Signature Validation Policy may be structured so that it can be computer processable. The current document includes, as an option, a formal structure for the signature validation policy based on the used of Abstract Syntax Notation 1 (ASN.1). Other formats of the signature validation policy are allowed by the present document. However, for a given signature policy there shall be one definitive form that has a unique binary encoded value.

The Signature Validation Policy includes rules regarding use of TSPs (CA, Attribute Authorities, Time Stamping Authorities) as well as rules defining the components of the electronic signature that shall be provided by the signer with data required by the verifier to provide long term proof.

5.2 Signed Information

The information being signed may be defined as a MIME-encapsulated message which can be used to signal the format of the content in order to select the right display or application. It can be composed of formatted text (e.g. EDIFACT), free text or of fields from an electronic form (e-form). For example, the Adobe™ format "pdf" may be used or the eXtensible Mark up Language (XML). Annex B defines how the content may be structured to indicate the type of signed data using MIME.

5.3 Components of an Electronic Signature

5.3.1 Reference to the Signature Policy

The definition of electronic signature includes: "a commitment has been **explicitly endorsed under a "Signature policy"**, at a given time, by a signer under an identifier, e.g. a name or a pseudonym, and optionally a role".

When two independent parties want to evaluate an electronic signature, it is fundamental that they get the same result. To meet this requirement the technical components and technical aspects used in creating the signature shall be referenced, this is provided by a reference to the "Signature Validation Policy". The "Signature Validation Policy" defines:

- the components of an electronic signature to be provided by the signer;
- any additional components (i.e. verifier components) used to validate an electronic signature at the time of receipt by a verifier and later by an arbitrator, auditor or other independent parties.

By signing over the signature policy identifier, the algorithm identifier and the hash of the signature policy, the signer explicitly indicates that he or she has applied the signature policy in creating the signature. Thus, undertakes any commitments implied by the signature policy, any indication of commitment type included in the electronic signature, and the user data that is signed.

The hash algorithm identifier and value is included to ensure that both the signer and verifier use exactly the same signature policy. This unambiguously binds the signer and verifier to same definitive form of the signature policy has a unique binary encoding.

In order to identify unambiguously the "Signature Validation Policy" to be used to verify the signature an identifier and hash of the "Signature policy" shall be part of the signed data. Additional information about the policy (e.g. web reference to the document) may be carried as "qualifiers" to the signature policy identifier.

5.3.2 Commitment Type Indication

The definition of electronic signature includes: "a commitment has been explicitly endorsed under a signature policy, at a given time, by a signer under an identifier, e.g. a name or a pseudonym, and optionally a role".

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 types indications shall be specified either as part of the signature policy or may be registered for generic use across multiple policies.

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

How commitment is indicated using the semantics of the data 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 which 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).

5.3.4 Certificate Identifier from the Signer

The definition of the ETSI electronic signature includes: "a commitment has been explicitly endorsed under a signature policy, at a given time, by a signer **under an identifier**, e.g. a name or a pseudonym, and optionally a role".

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

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

In order to counter this kind of attack, the identifier of the signer has to be protected by the digital signature from the signer.

Although it does not provide the same advantages as the previous technique, another technique to counter that threat has been identified. It requires all CAs to perform a Proof Of Possession of the private key at the time of registration. The problem with that technique is that it does not provide any guarantee at the time of verification and only some proof "after the event" may be obtained, if and only if the CA keeps the Proof Of Possession in an audit trail.

In order to identify unambiguously the certificate to be used for the verification of the signature an identifier of the certificate from the signer shall be part of the signed data.

5.3.5 Role Attributes

The definition of electronic signature includes: "a commitment has been explicitly endorsed under a non repudiation security policy, at a given time, by a signer **under** an identifier, e.g. a name or a pseudonym, and optionally **a role**".

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

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

- by placing a *claimed* role name in the CMS signed attributes field;
- by placing a attribute certificate containing a *certified* role name in the CMS signed attributes field.

NOTE: Another possible approach would have been to use additional attributes containing the roles name(s) in the signer's certificate. However, it was decided not to follow this approach as it breaks the basic philosophy of the certificate being issued for one primary purpose. Also, by using separate certificates for management of the signer's identity certificate and management of additional roles can simplify the management, as new identity keys need not be issued if a use of role is to be changed.

5.3.5.1 Claimed Role

The signer may be trusted to state his own role without any certificate to corroborate this claim. In which case the claimed role can be added to the signature as a signed attribute.

5.3.5.2 Certified Role

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 will be most of the time under the control of an organization or a company that is best placed to know which attributes are relevant for which individual. The Attribute Authority may use or point to public key certificates issued by any CA, provided that the appropriate trust may be placed in that CA. Attribute Certificates may have various periods of validity. That period may be quite short, e.g. one day. While this requires that a new Attribute Certificate is obtained every day, valid for that day, this can be advantageous since revocation of such certificates may not be needed. When signing, the signer will have to specify which Attribute Certificate it selects. 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 identify unambiguously the attribute certificate(s) to be used for the verification of the signature an identifier of the attribute certificate(s) from the signer shall be part of the signed data.

5.3.6 Signer Location

In some transactions the purported location of the signer at the time he or she applies his signature may need to be indicated. For this reason an optional location indicator shall be able to be included.

In order to provide indication of the location of the signer at the time he or she applied his signature a location attribute may be included in the signature.

5.3.7 Signing Time

The definition of electronic signature includes: "a commitment has been explicitly endorsed under a signature policy, **at a given time**, by a signer under an identifier, e.g. a name or a pseudonym, and optionally a role".

There are several ways to address this problem. The solution adopted in this document is to sign over a time which the signer claims is the signing time (i.e. claimed signing time) and to require a trusted time stamp to be obtained when building a ES with Timestamp. When a verifier accepts a signature, the two times shall be within acceptable limits.

The solution that is adopted in the present document offers the major advantage that electronic signatures can be generated without any on-line connection to a trusted time source (i.e. they may be generated off-line).

Thus two dates and two signatures are required:

- a signing time indicated by the signer and which is part of the data signed by the signer (i.e. part of the basic electronic signature);
- a time indicated by a TimeStamping Authority (TSA) which is signed over the digital signature value of the basic electronic signature. The signer, verifier or both may obtain the TSA timestamp.

In order for an electronic signature to be valid under a signature policy, it shall be timestamped by a TSA where the signing time as indicated by the signer and the time of time stamping as indicated by a TSA shall be "close enough" to meet the requirements of the signature validation policy.

"Close enough" means a few minutes, hours or even days according to the "Signature Validation Policy".

NOTE: The need for Timestamping is further explained in subclause 5.4.5.

A further optional attribute is defined in the present document to timestamp the content, to provide proof of the existence of the content, at the time indicated by the timestamp.

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 on-line connection to a trusted timestamping 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 timestamp (see subclause 8.12.4, Content Timestamp).

Also, the signing time should be between the time indicated by this timestamp and time indicated by the ES-T timestamp.

5.4 Components of Validation Data

5.4.1 Revocation Status Information

A verifier will have to prove that the certificate of the signer was valid at the time of the signature. This can be done by either:

- using Certificate Revocation Lists (CRLs);
- using responses from an on-line certificate status server (for example; obtained through the OCSP protocol).

5.4.2 CRL Information

When using CRLs to get revocation information, a verifier will have to make sure that he or she gets at the time of the first verification the appropriate certificate revocation information from the signer's CA. This should be done as soon as possible to minimize the time delay between the generation and verification of the signature. This involves checking that the signer certificate serial number is not included in the CRL. The signer, the verifier or any other third party may obtain either this CRL. If obtained by the signer, then it shall be conveyed to the verifier. It may be convenient to archive the CRL for ease of subsequent verification or arbitration. Alternatively, provided the CRL is archived elsewhere which is accessible for the purpose of arbitration, then the serial number of the CRL used may be archived together with the verified electronic signature.

It may happen that the certificate serial number appears in the CRL but with the status "suspended" (i.e. on hold). In such a case, the electronic signature is not yet valid, since it is not possible to know whether the certificate will or will not be revoked at the end of the suspension period. If a decision has to be taken immediately then the signature has to be considered as invalid. If a decision can wait until the end of the suspension period, then two cases are possible:

- the certificate serial number has disappeared from the list and thus the certificate can be considered as valid and that CRL shall be captured and archived either by the verifier or elsewhere and be kept accessible for the purpose of arbitration.
- the certificate serial number has been maintained on the list with the status definitively revoked and thus the electronic signature shall be considered as invalid and discarded.

At this point the verifier may be convinced that he or she got a valid signature, but is not yet in a position to prove at a later time that the signature was verified as valid. Before addressing this point, an alternative to CRL is to use OCSP responses.

5.4.3 OCSP Information

When using OCSP to get revocation information, a verifier will have to make sure that he or she gets at the time of the first verification an OCSP response that contains the status "valid". This should be done as soon as possible after the generation of the signature. The signer, the verifier or any other third party may fetch this OCSP response. Since OSCP responses are transient and thus are not archived by any TSP including CA, it is the responsibility of every verifier to make sure that it is stored in a safe place. The simplest way is to store them associated with the electronic signature. An alternative would be to store them in some storage so that they can then be easily retrieved.

In the same way as for the case of the CRL, it may happen that the certificate is declared as invalid but with the secondary status "suspended". In such a case, the electronic signature is not yet valid, since it is not possible to know whether the certificate will or will not be revoked at the end of the suspension period. If a decision has to be taken immediately then the electronic signature has to be considered as invalid. If a decision can wait until the end of the suspension period, then two cases are possible:

- an OCSP response with a valid status is obtained at a later date and thus the certificate can be considered as valid and that OCSP response shall be captured;
- an OCSP response with an invalid status is obtained with a secondary status indicating that the certificate is definitively revoked and thus the electronic signature shall be considered as invalid and discarded.

As in the CRL case, at this point, the verifier may be convinced that he or she got a valid signature, but is not yet in a position to prove at a later time that the signature was verified as valid.

5.4.4 Certification Path

A verifier will have to prove that the certification path was valid, at the time of the signature, up to a trust point according to the naming constraints and the certificate policy constraints from the "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 self-signed certificate from one trusted root of the "Signature Validation Policy". In addition, it will be necessary to capture the Authority Revocation Lists (ARLs) to prove than none of the CAs from the chain was revoked at the time of the signature.

As in the OCSP case, at this point, the verifier may be convinced that he or she got a valid signature, but is not yet in a position to prove at a later time that the signature was verified as valid.

5.4.5 Timestamping for Long Life of Signature

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

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

It would be quite unacceptable, to consider a signature as invalid even if the keys or certificates were later compromised. Thus there is a need to be able to demonstrate that the signature keys was valid around the time that the signature was created to provide long term evidence of the validity of a signature.

It could be the case that a certificate was valid at the time of the signature but revoked some time later. In this event, evidence shall be provided that the document was signed before the signing key was revoked. Timestamping by a Time Stamping Authority (TSA) can provide such evidence. A time stamp is obtained by sending the hash value of the given data to the TSA. The returned "timestamp" is a signed document that contains the hash value, the identity of the TSA, and the time of stamping. This proves that the given data existed *before* the time of stamping. Timestamping a digital signature (by sending a hash of the signature to the TSA) before the revocation of the signer's private key, provides evidence that the signature has been created before the key was revoked.

If a recipient wants to hold a valid *electronic* signature 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 timestamp 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 value of the signature. The time stamp can thus be provided by the signer together with the signed document, or obtained by the recipient following receipt of the signed document.

The time stamp is NOT a component of the Electronic Signature, but the essential component of the ES with Timestamp.

It is required in the present document that signer's digital signature value is timestamped by a trusted source, known as a TimeStamping Authority.

The present document requires that the signer's digital signature value is timestamped by a trusted source before the electronic signature can become a ES with Complete validation data (ES-C). The acceptable TSAs are specified in the Signature Validation Policy.

Should both the signer and verifier be required to timestamp the signature value to meet the requirements of the signature policy, the signature policy MAY specify a permitted time delay between the two time stamps.

5.4.6 Timestamping for Long Life of Signature before CA Key Compromises

Timestamped extended electronic signatures 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 a the time of the signature were valid, even in the case where one of the issuing keys or OCSP responder keys is later compromised.

The current document defines two ways of using timestamps to protect against this compromise:

- Timestamp the ES with Complete validation data, when an OCSP response is used to get the status of the certificate from the signer.
- Timestamp only the certification path and revocation information *references* when a CRL is used to get the status of the certificate from the signer.

NOTE: The signer, verifier or both may obtain the timestamp.

5.4.6.1 Timestamping the ES with Complete Validation Data

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 ES-C. Since the certification path and revocation information references are included in the ES with Complete validation data they are also protected. For the same cryptographic price, this provides an integrity mechanism over the ES with Complete validation data. Any modification can be immediately detected. It should be noticed that other means of protecting/detecting the integrity of the ES with Complete Validation Data exist and could be used.

Although the technique requires a time stamp for every signature, it is well suited for individual users wishing to have an integrity protected copy of all the validated signatures they have received.

By timestamping the complete electronic signature, including the digital signature as well as the references to the certificates and revocation status information used to support validation of that signature, the timestamp ensures that there is no ambiguity in the means of validating that signature.

This technique is referred to as ES with eXtended Validation data (ES-X), type 1 Timestamped in the present document.

NOTE: Trust is achieved in the references by including a hash of the data being referenced.

If it is desired for any reason to keep a copy of the additional data being referenced, the additional data may be attached to the electronic signature, in which case the electronic signature becomes a ES-X Long as defined by the present document.

A ES-X Long Timestamped is simply the concatenation of a ES-X Timestamped with a copy of the additional data being referenced.

5.4.6.2 Timestamping Certificates and Revocation Information References

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

Timestamping CA certificates will stop any attacker from issuing bogus CA certificates that could be claimed to existing before the CA key was compromised. Any bogus timestamped CA certificates will show that the certificate was created after the legitimate CA key was compromised. In the same way, timestamping CA CRLs, will stop any attacker from issuing bogus CA CRLs which could be claimed to existing before the CA key was compromised.

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

To comply with extended validation data, type 2 Timestamped, the present document requires the following:

- All the CA certificates references and revocation information references (i.e. CRLs) used in validating the ES-C are covered by one or more timestamp.

Thus a ES-C with a timestamp signature value at time T1, can be proved valid if all the CA and CRL references are timestamped at time T1+.

5.4.7 Timestamping for Long Life of Signature

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 probability.

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 cryptoanalysis, or improvements in cryptoanalytical techniques). Before this 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, a single technique, called Archive validation data, covering all the cases is being used in the present document.

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

The potential for Trusted Service Provider (TSP) key compromise should be significantly lower than 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 timestamps will protect against forgery. Each timestamp needs to be affixed before either the compromise of the signing key or of the cracking of the algorithms used by the TSA. TSAs (TimeStamping Authorities) should have long keys (e.g. which at the time of drafting the present document was 2048 bits for the signing RSA algorithm) and/or a "good" or different algorithm.

Nested timestamps 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.

5.4.8 Reference to Additional Data

Using type 1 or 2 of Timestamped extended validation data verifiers still needs to keep track of all the components that were used to validate the signature, in order to be able to retrieve them again later on. These components may be archived by an external source like a trusted service provider, in which case referenced information that is provided as part of the ES with Complete validation data (ES-C) is adequate. The actual certificates and CRL information reference in the ES-C can be gathered when needed for arbitration.

5.4.9 Timestamping for Mutual Recognition

In some business scenarios both the signer and the verifier need to timestamp their own copy of the signature value. Ideally the two timestamps should be as close as possible to each other.

Example: A contract is signed by two parties A and B representing their respective organizations, to timestamp the signer and verifier data two approaches are possible:

- under the terms of the contract pre-defined common "trusted" TSA may be used;
- if both organizations run their own timestamping services, A and B can have the transaction timestamped by these two timestamping services.

In the latter case, the electronic signature will only be considered as valid, if both timestamps were obtained in due time (i.e. there should not be a long delay between obtaining the two timestamps). Thus, neither A nor B can repudiate the signing time indicated by their own timestamping service. Therefore, A and B do not need to agree on a common "trusted" TSA to get a valid transaction.

It is important to note that signatures may be generated "off-line" and timestamped at a later time by anyone, e.g. by the signer or any recipient interested in validating the signature. The timestamp over the signature from the signer can thus be provided by the signer together with the signed document, and/or obtained by the verifier following receipt of the signed document.

The business scenarios may thus dictate that one or more of the long-term signature timestamping methods describe above be used. This will need to be part of a mutually agreed the Signature Validation Policy with is part of the overall signature policy under which digital signature may be used to support the business relationship between the two parties.

5.4.10 TSA Key Compromise

TSA servers should be built in such a way that once the private signature key is installed, that there is minimal likelihood of compromise over as long as possible period. Thus the validity period for the TSA's keys should be as long as possible.

Both the ES-T and the ES-C contain at least one time stamp over the signer's signature. In order to protect against the compromise of the private signature key used to produce that timestamp, the Archive validation data can be used when a different TimeStamping Authority key is involved to produce the additional timestamp. If it is believed that the TSA key used in providing an earlier timestamp may ever be compromised (e.g. outside its validity period), then the ES-A should be used. For extremely long periods this may be applied repeatedly using new TSA keys.

5.5 Multiple Signatures

Some electronic signatures may only be valid if they bear more than one signature. This is the case generally 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.

Several forms of multiple and counter signatures need to be supported, which fall into two basic categories:

- independent signatures;
- embedded signatures.

Independent signatures are parallel signatures where the ordering of the signatures is not important. The capability to have more than one independent signature over the same data shall be provided.

Embedded signatures are applied one after the other and are used where the order the signatures are applied is important. The capability to sign over signed data shall be provided.

These forms are described in subclause 8.13. 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 occurrence of the above two cases.

6 Signature Policy and Signature Validation Policy

The definition of electronic signature mentions: "a commitment has been **explicitly endorsed under a "Signature Policy"**, at a given time, by a signer under an identifier, e.g. a name or a pseudonym, and optionally a role".

Electronic signatures are commonly applied within the context of a legal or contractual framework. This establishes the requirements on the electronic signatures and any special semantics (e.g. agreement, intent). These requirements may be defined in very general abstract terms or in terms of detailed rules. The specific semantics associated with an electronic signature implied by a legal or contractual framework are outside the scope of the present document.

If the signature policy is recognized, within the legal/contractual context, as providing commitment, then the signer explicitly agrees with terms and conditions which are implicitly or explicitly part of the signed data.

When two independent parties want to evaluate an electronic signature, it is fundamental that they get the same result. It is therefore important that the conditions agreed by the signer at the time of signing are indicated to the verifier and any arbitrator. An aspect that enables this to be known by all parties is the signature policy. The technical implications of the signature policy on the electronic signature with all the validation data are called the "Signature Validation Policy". The signature validation policy specifies the rules used to validate the signature.

The present document does not mandate the form and encoding of the specification of the signature policy. However, for a given signature policy there shall be one definitive form that has a unique binary encoded value.

The present document includes, as an option, a formal structure for signature validation policy based on the use of Abstract Syntax Notation 1 (ASN.1).

Given the specification of the signature policy and its hash value an implementation of a verification process shall obey the rules defined in the specification.

The present document places no restriction on how it should be implemented. Provide the implementation conforms to the conformance requirements as define in subclauses 14.1, 14.2 and 14.3 implementation options include:

- A validation process that supports a specific signature policy as identified by the signature policy OID. Such an implementation should conform to a human readable description provided all the processing rules of the signature policy are clearly defined. However, if additional policies need to be supported, then such an implementation would need to be customized for each additional policy. This type of implementation may be simpler to implement initially, but can be difficult to enhance to support numerous additional signature policies.

- A validation process that is dynamically programmable and able to adapt its validation rules in accordance with a description of the signature policy provided in a computer-processable language. This present document defines such a policy using an ASN.1 structure (see subclause 11.1). This type of implementation could support multiple signature policies without being modified every time, provided all the validation rules specified as part of the signature policy are known by the implementation. (i.e. only requires modification if there are additional rules specified).

The precise content of a signature policy is not mandated by the current document. However, a signature policy shall be sufficiently definitive to avoid any ambiguity as to its implementation requirements. It shall be absolutely clear under which conditions an electronic signature should be accepted. For this reason, it should contain the following information:

- General information about the signature policy which includes:
  - a unique identifier of the policy;
  - the name of the issuer of the policy;
  - the date the policy was issued;
  - the field of application of the policy.

- The signature verification policy which includes:
  - the signing period,
  - a list of recognized commitment types;
  - rules for Use of Certification Authorities;
    - rules for Use of Revocation Status Information;
    - rules for Use of Roles;
    - rules for use of Timestamping and Timing;
    - signature verification data to be provided by the signer/collected by verifier;
    - any constraints on signature algorithms and key lengths.
- Other signature policy rules required to meet the objectives of the signature.

Variations of the validation policy rules may apply to different commitment types.

6.1 Identification of Signature Policy

When data is signed the signer indicates the signature policy applicable to that electronic signature by including an object identifier for the signature policy with the signature. The signer and verifier shall apply the rules specified by the identified policy. In addition to the identifier of the signature policy the signer shall include the hash of the signature policy, so it can be verified that the policy selected by the signer is the identical to the one being used the verifier.

A signature policy may be qualified by additional information. This can includes:

- A URL where a copy of the Signature Policy may be obtained;
- A user notice that should be displayed when the signature is verified.

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

A "Signature Policy" will be identifiable by an OID (Object Identifier) and verifiable using a hash of the signature policy.

6.2 General Signature Policy Information

General information should be recorded about the signature policy along with the definition of the rules which form the signature policy as described in subsequent subclauses. This should include:

- **Policy Object Identifier**: the "Signature Policy" will be identifiable by an OID (Object Identifier) whose last component (i.e. right most) is an integer that is specific to a particular version issued on the given date.
- **Date of issue:** when the "Signature Policy" was issued.
- **Signature Policy Issuer name**: an identifier for the body responsible for issuing the Signature Policy. This may be used by the signer or verifying in deciding if a policy is to be trusted, in which case the signer/verifier shall authenticate the origin of the signature policy as coming from the identified issuer.
- **Signing period**: the start time and date, optionally with an end time and date, for the period over which the signature policy may be used to generate electronic signatures.
- **Field of application**: this defines in general terms the general legal/contract/application contexts in which the signature policy is to be used and the specific purposes for which the electronic signature is to be applied.

6.3 Recognized Commitment Types

The signature validation policy may recognize one or more types of commitment as being supported by electronic signatures produced under the security policy.

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

Only recognized commitment types are allowed in an electronic signature.

The definition of a commitment type includes:

- the object identifier for the commitment;
- the contractual/legal/application context in which the signature may be used (e.g. submission of messages);
- a description of the support provided within the terms of the context (e.g. proof that the identified source submitted the message if the signature is created when message submission is initiated).

The definition of a commitment type can be registered:

- as part of the validation policy;
- as part of the application/contract/legal environment;
- as part of generic register of definitions.

The legal/contractual context will determine the rules applied to the signature, as defined by the signature policy and its recognized commitment types, make it fit for purpose intended.

6.4 Rules for Use of Certification Authorities

The certificate validation process of the verifier, and hence the certificates that may be used by the signer for a valid electronic signature, may be constrained by the combination of the trust point and certificate path constraints in the signature validation policy.

6.4.1 Trust Points

The signature validation policy defines the certification authority trust points that are to be used for signature verification. Several trust points may be specified under one signature policy. Specific trust points may be specified for a particular type of commitment defined under the signature policy. For a signature to be valid a certification path shall exists between the Certification Authority that has granted the certificate selected by the signer (i.e. the used user-certificate) and one of the trust point of the "Signature Validation Policy".

6.4.2 Certification Path

There may be constraints on the use of certificates issued by one or more CA(s) in the certificate chain and trust points. The two prime constraints are certificate policy constraints and naming constraints:

- Certificate policy constraints limit the certification chain between the user certificate and the certificate of the trusted point to a given set of certificate policies, or equivalents identified through certificate policy mapping.
- The naming constraints limit the forms of names that the CA is allowed to certify.

Name constraints are particularly important when a "Signature policy" identifies more than one trust point. In this case, a certificate of a particular trusted point may only be used to verify signatures from users with names permitted under the name constraint.

Certificate Authorities may be organized in a tree structure, this tree structure may represent the trust relationship between various CA(s) and the users CA. Alternatively, a mesh relationship may exist where a combination of tree and peer cross-certificates may be used. The requirement of the certificate path in the present document is that it provides the trust relationship between all the CAs and the signers user certificate. The starting point from a verification point of view, is the "trust point". A trust point is usually a CA that publishes self-certified certificates, is the starting point from which the verifier verifies the certificate chain. Naming constraints may apply from the trust point, in which case they apply throughout the set of certificates that make up the certificate path down to the signer's user certificate.

Policy constraints can be easier to process but to be effective require the presence of a certificate policy identifier in the certificates used in a certification path.

Certificate path processing, thus generally starts with one of the trust point from the signature policy and ends with the user certificate.

The certificate path processing procedures defined in RFC 2459 [7] clause 6 identifies the following initial parameters that are selected by the verifier in certificate path processing:

- acceptable certificate policies;
- naming constraints in terms of constrained and excluded naming subtree;
- requirements for explicit certificate policy indication and whether certificate policy mapping are allowed;
- restrictions on the certificate path length.

The signature validation policy identifies constraints on these parameters.

6.5 Revocation Rules

The signature policy should defines rules specifying requirements for the use of certificate revocation lists (CRLs) and/or on-line certificate status check service to check the validity of a certificate. These rules specify the mandated minimum checks that shall be carried out.

It is expected that in many cases either check may be selected with CRLs checks being carried out for certificate status that are unavailable from OCSP servers. The verifier may take into account information in the certificate in deciding how best to check the revocation status (e.g. a certificate extension field about authority information access or a CRL distribution point) provided that it does not conflict with the signature policy revocation rules.

6.6 Rules for the Use of Roles

Roles can be supported as claimed roles or as certified roles using Attribute Certificates.

6.6.1 Attribute Values

When signature under a role is mandated by the signature policy, then either Attribute Certificates may be used or the signer may provide a claimed role attribute. The acceptable attribute types or values may be dependent on the type of commitment. For example, a user may have several roles that allow the user to sign data that imply commitments based on one or more of his roles.

6.6.2 Trust Points for Certified Attributes

When a signature under a certified role is mandated by the signature policy, Attribute Authorities are used and need to be validated as part of the overall validation of the electronic signature. The trust points for Attribute Authorities do not need to be the same as the trust points to evaluate a certificate from the CA of the signer. Thus the trust point for verifying roles need not be the same as trust point used to validate the certificate path of the user's key.

Naming and certification policy constraints may apply to the AA in similar circumstance to when they apply to CA. Constraints on the AA and CA need not be exactly the same.

AA(s) may be used when a signer is creating a signature on behalf of an organization, they can be particularly useful when the signature represents an organizational role. AA(s) may or may not be the same authority as CA(s).

Thus, the Signature Policy identifies trust points that can be used for Attribute Authorities, either by reference to the same trust points as used for Certification Authorities, or by an independent list.

6.6.3 Certification Path for Certified Attributes

Attribute Authorities may be organized in a tree structure in similar way to CA where the AAs are the leafs of such a tree. Naming and other constraints may be required on attribute certificate paths in a similar manner to other electronic signature certificate paths.

Thus, the Signature Policy identify constraints on the following parameters used as input to the certificate path processing:

- acceptable certificate policies, including requirements for explicit certificate policy indication and whether certificate policy mapping is allowed;
- naming constraints in terms of constrained and excluded naming subtrees;
- restrictions on the certificate path length.

6.7 Rules for the Use of Timestamping and Timing

The following rules should be used when specifying, constraints on the certificate paths for timestamping authorities, constraints on the timestamping authority names and general timing constraints.

6.7.1 Trust Points and Certificate Paths

Signature keys from timestamping authorities will need to be supported by a certification path. The certification path used for timestamping authorities requires a trustpoint and possibly path constraints in the same way that the certificate path for the signer's key.

6.7.2 Timestamping Authority Names

Restrictions may need to be placed by the validation policy on the named entities that may act a timestamping authorities.

6.7.3 Timing Constraints - Caution Period

Before an electronic signature may really be valid, the verifier has to be sure that the holder of the private key was really the only one in possession of key at the time of signing. However, there is an inevitable delay between a compromise or loss of key being noted, and a report of revocation being distributed. To allow greater confidence in the validity of a signature, a "cautionary period" may be identified before a signature may be said to be valid with high confidence. A verifier may revalidate a signature after this cautionary signature, or wait for this period before validating a signature.

The validation policy may specify such a cautionary period.

6.7.4 Timing Constraints - Timestamp Delay

There will be some delay between the time that a signature is created and the time the signer's digital signature is timestamped. However, the longer this elapsed period the greater the risk of the signature being invalidated due to compromise or deliberate revocation of its private signing key by the signer. Thus the signature policy should specify a maximum acceptable delay between the signing time as claimed by the signer and the time included within the timestamp.

6.8 Rules for Verification Data to be followed

By specifying the requirements on the signer and verifier the responsibilities of the two parties can be clearly defined to establish all the necessary information.

These verification data rules should include:

- requirements on the signer to provide given signed attributes;
- requirements on the verifier to obtain additional certificates, CRLs, results of on line certificate status checks and to use timestamps (if no already provided by the signer).

6.9 Rules for Algorithm Constraints and Key Lengths

The signature validation policy may identify a set of signing algorithms (hashing, public key, combinations) and minimum key lengths that may be used:

- by the signer in creating the signature;
- in end entity public key Certificates;
- CA Certificates;
- attribute Certificates;
- by the timestamping authority.

6.10 Other Signature Policy Rules

The signature policy may specify additional policy rules, for example rules that relate to the environment used by the signer. These additional rules may be defined in computer processable and/or human readable form.

6.11 Signature Policy Protection

When signer or verifier obtains a copy of the Signature Policy from an issuer, the source should be authenticated (for example by using electronic signatures).

When the signer references a signature policy the Object Identifier (OID) of the policy, the hash value and the hash algorithm OID of that policy shall be included in the Electronic Signature.

It is a mandatory requirement of this present document that the signature policy value computes to one, and only one hash value using the specified hash algorithm. This means that there shall be a single binary value of the encoded form of the signature policy for the unique hash value to be calculated. For example, there may exist a particular file type, length and format on which the hash value is calculated which is fixed and definitive for a particular signature policy.

The hash value may be obtained by:

- the signer performing his own computation of the hash over the signature policy using his preferred hash algorithm permitted by the signature policy, and the definitive binary encoded form;

the signer, having verified the source of the policy, may use both the hash algorithm and the hash value included in the computer processable form of the policy (see subclause 11.1).

7 Identifiers and roles

7.1 Signer Name Forms

The name used by the signer, held as the subject in the signer's certificate, shall uniquely identify the entity. The name shall be allocated and verified on registration with the Certification Authority, either directly or indirectly through a Registration Authority, before being issued with a Certificate.

The present document places no restrictions on the form of the name. The subject's name may be a distinguished name, as defined in ITU-T Recommendation X.500 [15], held in the subject field of the certificate, or any other name form held in the X.509 subjectAltName certificate extension field. In the case that the subject has no distinguished name, the subject name can be an empty sequence and the subjectAltName extension shall be critical.

Further guidance on naming individual citizens and individuals within an organization is given in annex F.

7.2 TSP Name Forms

All TSP name forms (Certification Authorities, Attribute Authorities and TimeStamping Authorities) shall be in the form of a distinguished name held in the subject field of the certificate.

The TSP name form shall include identifiers for the organization providing the service and the legal jurisdiction (e.g. country) under which it operates.

7.3 Roles and Signer Attributes

Where a signer signs as an individual but wishes to also identify him/herself as acting on behalf of an organization, it may be necessary to provide two independent forms of identification. The first identity, with is directly associated with the signing key identifies him/her as an individual. The second, which is managed independently, identifies that person acting as part of the organization, possibly with a given role.

In this case the first identity is carried in the subject/subjectAltName field of the signer's certificate as described above.

The present document supports the following means of providing a second form of identification:

- by placing a secondary name field containing a *claimed* role in the CMS signed attributes field;
- by placing an attribute certificate containing a *certified* role in the CMS signed attributes field.

8 Data structure of an Electronic Signature

This clause builds upon the existing Crypographic Message Syntax (CMS), as defined in RFC 2630 [9], and Enhanced Security Services (ESS), as defined in RFC 2634 [10]. The overall structure of Electronic Signature is as defined in CMS. The Electronic Signature (ES) uses attributes defined in CMS, ESS and this present document. This present document defines ES attributes not defined elsewhere.

The mandated set of attributes and the digital signature value is defined as the minimum Electronic Signature (ES) required by the present document. A signature policy MAY mandate other signed attributes are present.

NOTE: This clause duplicates some text and ASN.1 data structures that may be found in other standards, in particular CMS (RFC 2630 [9]) and ESS (RFC 2634 [10]). All duplicated text and ASN.1 is provided in the present documents for completeness and to avoid multiple reference to other documents with the aim of making it easier to read. Profiling the imported data structures to meet specific requirements of the present document provides value added. Copyright of the CMS and ESS text remains with the IETF.

8.1 General Syntax

The Cryptographic Message Syntax (CMS) associates a content type identifier with a content. The syntax shall have ASN.1 type ContentInfo:

```
ContentInfo ::= SEQUENCE {
  contentType ContentType,
  content [0] EXPLICIT ANY DEFINED BY contentType }

ContentType ::= OBJECT IDENTIFIER
```

The fields of ContentInfo have the following meanings:

- `contentType` indicates the type of the associated content. It is an object identifier; it is a unique string of integers assigned by an authority that defines the content type.
- `content` is the associated content. The type of content can be determined uniquely by contentType. The present document supports Content types for data and signed-data.

8.2 Data Content Type

The following object identifier identifies the data content type:

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

The data content type is intended to refer to arbitrary octet strings, such as ASCII text files; the interpretation is left to the application. Such strings need not have any internal structure (although they could have their own ASN.1 definition or other structure). The data content type is encapsulated in the signed-data content type.

8.3 Signed-data Content Type

The signed-data content type consists of a content of any type and zero or more signature values. Any number of signers in parallel can sign any type of content.

The typical application of the signed-data content type represents one signer's digital signature on content of the data content type.

The process by which signed-data is constructed involves the following steps:

- For each signer, a message digest, or hash value, is computed on the content with a signer-specific message-digest algorithm. If the signer is signing any information other than the content, the message digest of the content and the other information are digested with the signer's message digest (see signed attributes), and the result becomes the "message digest".
- For each signer, the message digest is digitally signed using the signer's private key.
- For each signer, the signature value and other signer-specific information are collected into a SignerInfo value. Certificates and CRLs for each signer, and those not corresponding to any signer, are collected in this step.
- The message digest algorithms for all the signers and the SignerInfo values for all the signers are collected together with the content into a SignedData value.

A recipient independently computes the message digest. This message digest and the signer's public key are used to verify the signature value. The signer's public key is referenced either by an issuer distinguished name along with an issuer-specific serial number or by a subject key identifier that uniquely identifies the certificate containing the public key. To make sure that the verifier uses the right signers key, the present document mandates that the hash of the signers certificate is always included in the Signing Certificate signed attribute (see subclause 8.1).

8.4 SignedData Type

The following object identifier identifies the signed-data content type:

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

The signed-data content type shall have ASN.1 type **SignedData**:

```
      SignedData ::= SEQUENCE {
        version CMSVersion,
        digestAlgorithms DigestAlgorithmIdentifiers,
        encapContentInfo EncapsulatedContentInfo,
        certificates [0] IMPLICIT CertificateSet OPTIONAL,
        crls [1] IMPLICIT CertificateRevocationLists OPTIONAL,
        signerInfos SignerInfos }

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo

DigestAlgorithmIdentifier ::= AlgorithmIdentifier -- AS defined in X.509

CMSVersion ::= INTEGER  { v0(0), v1(1), v2(2), v3(3), v4(4) }

CertificateChoices ::= CHOICE {
  certificate Certificate,  -- See X.509
  extendedCertificate [0] IMPLICIT ExtendedCertificate,  -- Obsolete, not used
  attrCert [1] IMPLICIT AttributeCertificate }  -- See X.509 & X9.57

CertificateSet ::= SET OF CertificateChoices

UnauthAttributes ::= UnsignedAttributes

CertificateRevocationLists ::= SET OF CertificateList -- See X.509

SubjectKeyIdentifier ::= OCTET STRING

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier
```

The fields of type **SignedData** have the following meanings within the context of the present document:

version is the syntax version number. The value of version shall be 3.

digestAlgorithms is a collection of message digest algorithm identifiers. There may be any number of elements in the collection, including zero. Each element identifies the message digest algorithm, along with any associated parameters, used by one or more signer. The collection is intended to list the message digest algorithms employed by all of the signers, in any order, to facilitate one-pass signature verification. The message digesting process is described in subclause 8.6. A list of digest algorithms, including those required by RFC 2630 [9], is given in annex E.

encapContentInfo is the signed content, consisting of a content type identifier and the content itself.

certificates is a collection of certificates. It is intended that the set of certificates be sufficient to contain chains from a recognized "root" or "top-level certification authority" to all of the signers in the **signerInfos** field. There may be more certificates than necessary, and there may be certificates sufficient to contain chains from two or more independent top-level certification authorities. There may also be fewer certificates than necessary, if it is expected that recipients have an alternate means of obtaining necessary certificates (e.g. from a previous set of certificates). The identification of signer's certificate used to create the signature is always signed (see subclause 8.1). The validation policy may specify requirements for the presence of certain certificates.

crls is a collection of certificate revocation lists (CRLs). It is intended that the set contain information sufficient to determine whether or not the certificates in the certificates field are valid, but such correspondence is not necessary. There may be more CRLs than necessary, and there may also be fewer CRLs than necessary.

signerInfos is the collection of per-signer information. There may be any number of elements in the collection, including zero. The details of the **SignerInfo** type are discussed later.

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

8.5 EncapsulatedContentInfo Type

The content is represented in the type **EncapsulatedContentInfo**:

```
EncapsulatedContentInfo ::= SEQUENCE {
  eContentType ContentType,
  eContent [0] EXPLICIT OCTET STRING OPTIONAL }

ContentType ::= OBJECT IDENTIFIER
```

The fields of type **EncapsulatedContentInfo** have the following meanings:

- **eContentType** is an object identifier that uniquely specifies the content type (e.g. id-data).
- **eContent** is the content itself, carried as an octet string. The **eContent** need not be DER encoded.

The optional omission of the **eContent** within the **EncapsulatedContentInfo** field makes it possible to construct "external signatures". In the case of external signatures, the content being signed is absent from the **EncapsulatedContentInfo** value included in the signed-data content type. If the **eContent** value within **EncapsulatedContentInfo** is absent, then the signatureValue is calculated and the **eContentType** is assigned as though the **eContent** value was present.

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 which is signed is archived in such as way as to preserve the 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.

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

8.6 SignerInfo Type

Per-signer information is represented in the type **SignerInfo.** In the case of multiple independent signatures (see subclause 5.6), there is an instance of this field for each signer.

The syntax of SignerInfo is as follows:

```
SignerInfo ::= SEQUENCE {
  version CMSVersion,
  sid SignerIdentifier,
  digestAlgorithm DigestAlgorithmIdentifier,
  signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
  signatureAlgorithm SignatureAlgorithmIdentifier,
  signature SignatureValue,
  unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL }

SignerIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  subjectKeyIdentifier [0] SubjectKeyIdentifier }

SignedAttributes ::= SET SIZE (1..MAX) OF Attribute

UnsignedAttributes ::= SET SIZE (1..MAX) OF Attribute

Attribute ::= SEQUENCE {
  attrType OBJECT IDENTIFIER,
  attrValues SET OF AttributeValue }

AttributeValue ::= ANY

SignatureValue ::= OCTET STRING
```

The fields of type **SignerInfo** have the following meanings:

version is the syntax version number. If the **SignerIdentifier** is the `CHOICE` **issuerAndSerialNumber**, then the version shall be 1. If the **SignerIdentifier** is **subjectKeyIdentifier**, then the version shall be 3.

sid specifies the signer's certificate (and thereby the signer's public key). The signer's public key is needed by the recipient to validate the signature. **SignerIdentifier** provides two alternatives for specifying the signer's public key. The **issuerAndSerialNumber** alternative identifies the signer's certificate by the issuer's distinguished name and the certificate serial number; the **subjectKeyIdentifier** identifies the signer's certificate by the X.509 **subjectKeyIdentifier** extension value.

digestAlgorithm identifies the message digest algorithm, and any associated parameters, used by the signer. The message digest is computed on either the content being signed or the content together with the signed attributes. The message digest algorithm should be among those listed in the **digestAlgorithms** field of the associated **SignerData** A list of digest algorithms including those required by RFC 2630 [9] is given in annex E.

signedAttributes is a collection of attributes that are signed. The field is optional, but it shall be present if the content type of the **EncapsulatedContentInfo** value being signed is not id-data.

Each **SignedAttribute** in the SET shall be DER encoded. The present document mandates this field shall, as a minimum, contain the following attributes:

- **ContentType** as defined in subclause 8.7.1.
- **MessageDigest** as defined in subclause 8.7.2.
- **SigningTime** as defined in subclause 8.7.3.
- **SigningCertificate** as defined in subclause 8.8.1.
- **SignaturePolicyId** as defined in subclause 8.9.1.

signatureAlgorithm identifies the signature algorithm, and any associated parameters, used by the signer to generate the digital signature.

signature is the result of digital signature generation, using the message digest and the signer's private key.

unsignedAttributes is a collection of attributes that are not signed. The field is optional. Useful attribute types, such as countersignatures.

The fields of type **SignedAttribute** and **UnsignedAttribute** have the following meanings:

attrType indicates the type of attribute. It is an object identifier.

attrValues is a set of values that comprise the attribute. The type of each value in the set can be determined uniquely by **attrType**.

8.6.1 Message Digest Calculation Process

The message digest calculation process computes a message digest on either the content being signed or the content together with the signed attributes. In either case, the initial input to the message digest calculation process is the "value" of the encapsulated content being signed. Specifically, the initial input is the **encapContentInfo eContent** OCTET STRING to which the signing process is applied. Only the octets comprising the value of the **eContent** OCTET STRING are input to the message digest algorithm, not the tag or the length octets.

The result of the message digest calculation process depends on whether the **signedAttributes** field is present. When the field is absent, the result is just the message digest of the content as described above. When the field is present, however, the result is the message digest of the complete DER encoding of the **SignedAttributes** value contained in the **signedAttributes** field. Since the **SignedAttributes** value, when present, shall contain the content type and the content message digest attributes, those values are indirectly included in the result. The content type attribute is not required when used as part of a countersignature unsigned attribute. A separate encoding of the signedAttributes field is performed for message digest calculation. The IMPLICIT [0] tag in the signedAttributes field is not used for the DER encoding, rather an EXPLICIT SET OF tag is used. That is, the DER encoding of the SET OF tag, rather than of the IMPLICIT [0] tag, is to be included in the message digest calculation along with the length and content octets of the SignedAttributes value.

When the **signedAttributes** field is absent, then only the octets comprising the value of the **signedData encapContentInfo eContent** OCTET STRING (e.g., the contents of a file) are input to the message digest calculation. This has the advantage that the length of the content being signed need not be known in advance of the signature generation process.

8.6.2 Message Signature Generation Process

The input to the digital signature generation process includes the result of the message digest calculation process and the signer's private key. The details of the digital signature generation depend on the signature algorithm employed. The object identifier, along with any parameters, that specifies the signature algorithm employed by the signer is carried in the **signatureAlgorithm** field. The signature value generated by the signer is encoded as an OCTET STRING and carried in the signature field.

8.6.3 Message Signature Verification Process

The procedures for CMS signed data validation are specified in CMS and repeated in clause 9. The input to the signature verification process includes the result of the message digest calculation process and the signer's public key which verified as correct using the **ESS or Other signing certificate** attribute. The recipient may obtain the correct public key for the signer by any means, but the preferred method is from a certificate obtained from the **SignedData** certificates field and then validated using the hash of the signer's certificate in the **signing certificate** attribute. The validation of the signer's public key may be based on certification path processing (see RFC 2459 [7]). The details of the signature verification depend on the signature algorithm employed.

The recipient may not rely on any message digest values computed by the originator. If the **signedData signerInfo** includes **signedAttributes**, then the content message digest shall be calculated over all the **signedAttributes**. For the signature to be valid, the message digest value calculated by the recipient shall be the same as the value of the **messageDigest** attribute included in the **signedAttributes** of the **signedData signerInfo**.

8.7 CMS Imported Mandatory Present Attributes

The following attributes SHALL be present with the signed-data defined by the present document. The attributes are defined in CMS and are imported into the present document. The full definitions are repeated in this clause for completeness and to avoid multiple reference to other documents. Profiling the imported attributes provides value added.

8.7.1 Content Type

The content-type attribute type specifies the content type of the **ContentInfo** value being signed in signed-data. The content-type attribute type is always required by the present document. The content-type attribute shall be a signed attribute.

The following object identifier identifies the content-type attribute:

```
id-contentType OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) 3 }
```

Content-type attribute values have ASN.1 type **ContentType**:

```
ContentType ::= OBJECT IDENTIFIER
```

A content-type attribute shall have a single attribute value, even though the syntax is defined as a SET OF **AttributeValue**. There shall not be zero or multiple instances of **AttributeValue** present.

The **SignedAttributes** syntaxes are each defined as a SET OF Attributes. The **SignedAttributes** in a **signerInfo** shall not include multiple instances of the content-type attribute.

8.7.2 Message Digest

The message-digest attribute type specifies the message digest of the **encapContentInfo eContent** OCTET STRING being signed in signed-data. For signed-data, the message digest is computed using the signer's message digest algorithm. Within signed-data, the message-digest signed attribute type is always required by the present document. The message-digest attribute shall be a signed attribute.

The following object identifier identifies the message-digest attribute:

```
id-messageDigest OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) 4 }
```

Message-digest attribute values have ASN.1 type **MessageDigest**:

```
MessageDigest ::= OCTET STRING
```

A message-digest attribute shall have a single attribute value, even though the syntax is defined as a SET OF **AttributeValue**. There shall not be zero or multiple instances of **AttributeValue** present. The **SignedAttributes** syntax is defined as a SET OF Attributes. The **SignedAttributes** in a **signerInfo** shall not include multiple instances of the message-digest attribute.

8.7.3 Signing Time

The signing-time attribute type specifies the time at which the signer claims to have performed the signing process.

The signing-time attribute is a signed attribute. The present document mandates the signing-time attribute shall be present in the **SignedData**. The signing-time attribute shall be a signed attribute

The following object identifier identifies the signing-time attribute:

```
id-signingTime OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) 5 }
```

Signing-time attribute values have ASN.1 type **SigningTime**:

```
SigningTime ::= Time

Time ::= CHOICE {
  utcTime          UTCTime,
  generalizedTime  GeneralizedTime }
```

The definition of Time matches the one specified in the 1997 version of ITU-T Recommendation X.509 [1]. Dates through the year 2049 maybe encoded as **UTCTime**, and dates in the year 2050 or later shall be encoded as **GeneralizedTime**.

This present document recommends the use of **GeneralizedTime**.

A signing-time attribute shall have a single attribute value, even though the syntax is defined as a SET OF AttributeValue. There shall not be zero or multiple instances of **AttributeValue** present.

The **SignedAttributes** syntax is defined as a SET OF Attributes. The **SignedAttributes** in a **signerInfo** shall not include multiple instances of the signing-time attribute.

8.8 Alternative Signing Certificate Attributes

One, and only one, of the following two alternative attributes SHALL be present with the signed-data defined by the present document to identify the signing certificate. Both attributes include an identifier and a hash of the signing certificate. The first, which is adopted in existing standards, may be used if with the SHA-1 hashing algorithm. The other shall be used for other hashing algorithms are to be supported.

The signing certificate attribute is designed to prevent the simple substitution and re-issue attacks, and to allow for a restricted set of authorization certificates to be used in verifying a signature.

8.8.1 ESS Signing Certificate Attribute Definition

This attributes is as defined in RFC 2634 [10] (Enhanced Security Services for S/MIME) and is imported into this ETSI specification. The full definitions are repeated in this clause for completeness and to avoid multiple reference to other documents. Profiling the imported attributes provides value added.

The ESS signing certificate attribute shall be a signed attribute.

The following object identifier identifies the signing certificate attribute:

```
id-aa-signingCertificate OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-aa(2) 12 }
```

The signing certificate attribute value has the ASN.1 syntax **SigningCertificate**

```
SigningCertificate ::=  SEQUENCE {
    certs        SEQUENCE OF ESSCertID,
    policies     SEQUENCE OF PolicyInformation OPTIONAL
                 -- NOT USED IN THE PRESENT DOCUMENT
    }
```

The first certificate identified in the sequence of certificate identifiers shall be the certificate used to verify the signature. The present document mandates the presence of this attribute as a signed CMS attribute, and the sequence shall not be empty. The certificate used to verify the signature shall be identified in the sequence, the Signature Validation Policy may mandate other certificates be present, that may include all the certificates up to the point of trust.

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

The **issuerAndSerialNumber** will be present in the **SignerInfo**, it shall be consistent with **issuerSerial** field. The certificate identified 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.

The sequence of policy information terms identifies those certificate policies that the signer asserts apply to the certificate, and under which the certificate should be relied upon. This value suggests a policy value to be used in the relying party's certification path processing. This field is not used in the present document.

The **SigningCertificate** attribute shall be a signed attribute. CMS defines **SignedAttributes** as a SET OF Attribute. A **SignerInfo** shall not include multiple instances of the **SigningCertificate** attribute. CMS defines the ASN.1 syntax for the s**ignedattributes** to include **attrValues** SET OF **AttributeValue**. A **SigningCertificate** attribute shall include only a single instance of **AttributeValue**. There shall not be zero or multiple instances of **AttributeValue** present in the **attrValues** SET OF **AttributeValue**.

NOTE: Where an attribute certificate is used by the signer to associate a role, or other attributes of the signer, with the electronic signature this is placed in the Signer Attribute attribute as defined in subclause 8.12.3.

8.8.1.1 Certificate Identification

The issuer/serial number pair is therefore sufficient to identify the correct signing certificate. This information is already present, as part of the **SignerInfo** object, and duplication of this information would be unfortunate. A hash of the entire certificate serves the same function (allowing the receiver to verify that the same certificate is being used as when the message was signed), is smaller, and permits a detection of the simple substitution attacks.

Attribute certificates and additional public key certificates containing authorization information do not have an issuer/serial number pair represented anywhere in a **SignerInfo** object. When an **attributecertificate** or an additional public key certificate is not included in the **SignedData** object, it becomes much more difficult to get the correct set of certificates based only on a hash of the certificate. For this reason, these certificates SHOULD be identified by the **IssuerSerial** field in ESSCertID.

The present document defines a certificate identifier as:

```
ESSCertID ::=  SEQUENCE {
     certHash                 Hash,
     issuerSerial             IssuerSerial OPTIONAL }

Hash ::= OCTET STRING -- SHA1 hash of entire certificate
```

```
IssuerSerial ::= SEQUENCE {
     issuer                   GeneralNames,
     serialNumber             CertificateSerialNumber }
```

When creating an **ESSCertID**, the **certHash** is computed over the entire DER encoded certificate including the signature.

The **issuerSerial** would normally be present unless the value can be inferred from other information.

When encoding **IssuerSerial**, **serialNumber** is the serial number that uniquely identifies the certificate. For non-attribute certificates, the issuer shall contain only the issuer name from the certificate encoded in the **directoryNam**e choice of **GeneralNames**. For attribute certificates, the issuer shall contain the issuer name field from the attribute certificate.

8.8.2 Other Signing Certificate Attribute Definition

The following attribute is identical to the ESS SigningCertificate defined above except that this attribute can be used with hashing algorithms other than SHA-1.

This attribute shall be used in the same manner as defined above for the ESS SigningCertificate attribute.

The following object identifier identifies the signing certificate 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 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 ::= 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 }
```

8.9 Additional Mandatory Attributes

8.9.1 Signature policy Identifier

The present document mandates that a reference to the signature policy, which defines the rules for creation and validation of an electronic signature, is included as a signed attribute with every signature. The signature policy identifier shall be a signed attribute.

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 ::= SEQUENCE {
        sigPolicyIdentifier   SigPolicyId,
        sigPolicyHash         SigPolicyHash,
```

```
        sigPolicyQualifiers   SEQUENCE SIZE (1..MAX) OF
                                SigPolicyQualifierInfo OPTIONAL}
```

The sigPolicyIdentifier field contains an object-identifier which uniquely identifies a specific version of the signature policy. The syntax of this field is as follows:

```
   SigPolicyId ::= OBJECT IDENTIFIER
```

The sigPolicyHash field contains the identifier of the hash algorithm and the hash of the value of the signature policy.

If the signature policy is defined using ASN.1 (see 11.1) the hash is calculated on the value without the outer type and length fields and the hashing algorithm shall be as specified in the field signPolicyHshAlg.

If the signature policy is defined using another structure, the type of structure and the hashing algorithm shall be either specified as part of the signature policy, or indicated using a signature policy qualifier.

```
   SigPolicyHash ::= ETSIHashAlgAndValue
```

A signature policy identifier may be qualified with other information about the qualifier. The semantics and syntax of the qualifier is as associated with 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 web URI or URL reference to the signature policy;
- **spUserNotice**: this contains a user notice which should be displayed whenever the signature is validated.

```
-- 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)) }
```

8.10 CMS Imported Optional Attributes

The following attributes MAY be present with the signed-data defined by the present document. The attributes are defined in CMS and are imported into this ETSI specification. The full definitions are repeated in this clause for completeness and to avoid multiple reference to other documents. Value added is provided by profiling the imported attributes.

8.10.1 Countersignature

The countersignature attribute type specifies one or more signatures on the content octets of the DER encoding of the **signatureValue** field of a **SignerInfo** value in signed-data. Thus, the countersignature attribute type countersigns (signs in serial) another signature.

The countersignature attribute shall be an unsigned attribute; it cannot be a signed attribute, an authenticated attribute, or an unauthenticated attribute.

The following object identifier identifies the countersignature attribute:

```
id-countersignature OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) 6 }
```

Countersignature attribute values have ASN.1 type **Countersignature**:

```
Countersignature ::= SignerInfo
```

Countersignature values have the same meaning as **SignerInfo** values for ordinary signatures, except that:

- The signedAttributes field shall contain a message-digest attribute if it contains any other attributes, but need not contain a content-type attribute, as there is no content type for **countersignatures**.
- The input to the message-digesting process is the contents octets of the DER encoding of the **signatureValue** field of the **SignerInfo** value with which the attribute is associated.

A **countersignature** attribute can have multiple attribute values. The syntax is defined as a SET OF AttributeValue, and there shall be one or more instances of AttributeValue present. A countersignature shall be an **UnsignedAttribute**.

The **UnsignedAttributes** syntax is defined as a SET OF Attributes. The **UnsignedAttributes** in a **signerInfo** may include multiple instances of the countersignature attribute.

A **countersignature**, since it has type **SignerInfo**, can itself contain a **countersignature** attribute. Thus it is possible to construct arbitrarily long series of **countersignatures**.

8.11 ESS Imported Optional Attributes

The following attributes MAY be present with the signed-data defined by the present document. The attributes are defined in ESS and are imported into this ETSI specification. The full definitions are repeated in this clause for completeness and to avoid multiple reference to other documents. Value added is provided by profiling the imported attributes.

8.11.1 Signed Content Reference 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. The first **SignedData** shall include a content Identifier signed attribute, which SHOULD be constructed as specified in RFC 2643 subclause 2.7. The second **SignedData** links to the first by including a Content Reference signed attribute containing the content type, content identifier, and signature value from the first **SignedData**. The content reference shall be a signed attribute.

The following object identifier identifies the content reference attribute:

```
id-aa-contentReference   OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 10 }
```

The content reference attribute values have ASN.1 type **ContentReference**:

```
ContentReference ::= SEQUENCE {
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue OCTET STRING }
```

8.11.2 Content Identifier Attribute

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

The following object identifier identifies the content identifier attribute:

```
id-aa-contentIdentifier OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 7}
```

The content identifier attribute values have ASN.1 type **ContentIdentifier**:

```
ContentIdentifier ::= OCTET STRING
```

The minimal signedContentIdentifier 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.

8.12 Additional Optional Attributes

8.12.1 Commitment Type Indication Attribute

There may be situation were 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 **commitmentTypeIndication** attribute conveys such information.

The **commitmentTypeIndication** attribute shall be a signed attribute.

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;
- 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 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 this field.

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 }
```

The use of any qualifiers to the commitment type 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 meaning:

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).

NOTE: See clause A.3 for a full description of the commitment types defined above.

8.12.2 Signer Location

The signer-location attribute is an attribute which specifies a mnemonic for an address associated with the signer at a particular geographical (e.g. city) location. The mnemonic is registered in the country in which the signer is located and is used in the provision of the Public Telegram Service (according to ITU-T Recommendation F.1 [5]).

The signer-location attribute shall be a signed attribute.

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
```

8.12.3 Signer Attributes

The signer-attributes attribute is an attribute which specifies additional attributes of the signer (e.g. role).

It may be either:

- claimed attributes of the signer;
- certified attributes of the signer;
- the signer-attribute attribute shall be a signed attribute attributes.

The signer-attributes attribute shall be a signed attribute.

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-attribute attribute 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 X.509 : see section 10.3
```

NOTE: The claimed and certified attribute are imported from ITU-T Recommendations X.501 [16] and ITU-T Recommendation X.509: "Draft Amendment on Certificate Extensions", October 1999.

8.12.4 Content Timestamp

The content timestamp attribute is an attribute which is the timestamp of the signed data content before it is signed.

The content timestamp attribute shall be a signed attribute.

The following object identifier identifies the signer-attribute 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 timestamp attribute values have ASN.1 type ContentTimestamp:

```
ContentTimestamp::= TimeStampToken
```

The value of messageImprint field within **TimeStampToken** shall be a hash of the value of **eContent** field within **encapContentInfo** within the **signedData**.

For further information and definition of TimeStampToken see subclause 10.4.

8.13 Support for Multiple Signatures

8.13.1 Independent Signatures

Multiple independent signatures (see subclause 5.5) are supported by independent **SignerInfo** from each signer.

Each **SignerInfo** shall include all the attributes required under the present document and shall be processed independently by the verifier.

8.13.2 Embedded Signatures

Multiple embedded signatures (see subclause 5.6) are supported using the counter-signature unsigned attribute (see subclause 10.1). Each counter signature is carried in **Countersignature** held as an unsigned attribute to the **SignerInfo** to which the counter-signature is applied.

9 Validation Data

This clause specifies the validation data structures which builds on the electronic signature specified in clause 8. This includes:

- **Timestamp** applied to the electronic signature value.
- **Complete validation data** which comprises the timestamp of the signature value, plus references to all the certificates and revocation information used for full validation of the electronic signature.

The following optional eXtended forms of validation data are also defined:

- **X-timestamp**: there are two types of timestamp used in extended validation data defined by the present document.
  - Type 1 -Timestamp which comprises a timestamp over the ES with Complete validation data (ES-C);
  - Type 2 X-Timestamp which comprises of a timestamp over the certification path references and the revocation information references used to support the ES-C.
- **X-Long:** this comprises a Complete validation data plus the actual values of all the certificates and revocation information used in the ES-C.
- **X-Long-Timestamp**: this comprises a Type 1 or Type 2 X-Timestamp plus the actual values of all the certificates and revocation information used in the ES-C.

This clause also specifies the data structures used in Archive validation data:

- Archive validation data comprises a Complete validation data, the certificate and revocation values (as in a X-Long validation data), any other existing X-timestamps, plus the Signed User data and an additional archive timestamp over all that data. An archive timestamp may be repeatedly applied after long periods to maintain validity when electronic signature and timestamping algorithms weaken.

The additional data required to create the forms of electronic signature identified above is carried as unsigned attributes associated with an individual signature by being placed in the **unsignedAttrs** field of **SignerInfo** (see clause 6). Thus all the attributes defined in clause 9 are unsigned attributes.

NOTE: Where multiple signatures are to be supported, as described in subclause 8.13, each signature has a separate **SignerInfo**. Thus, each signature requires its own unsigned attribute values to create ES-T, ES-C etc.

9.1 Electronic Signature Timestamp

An Electronic Signature with Timestamp is an Electronic Signature for which part, but not all, of the additional data required for validation is available (i.e. some certificates and revocation information is available but not all).

The minimum structure Timestamp validation data is:

- The Signature Timestamp Attribute as defined in subclause 9.1.1 over the ES signature value.

9.1.1 Signature Timestamp Attribute Definition

The Signature Timestamp attribute is timestamp of the signature value. It is an unsigned attribute. Several instances of this attribute may occur with an electronic signature, from different TSAs.

The Signature Validation Policy specifies, in the **signatureTimestampDelay** field of **TimestampTrustConditions,** an maximum acceptable time difference which is allowed between the time indicated in the signing time attribute and the time indicated by the Signature Timestamp attribute. If this delay is exceeded then the electronic signature shall be considered as invalid.

The following object identifier identifies the 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}
```

The Signature timestamp attribute value has ASN.1 type **SignatureTimeStampToken**:

```
SignatureTimeStampToken ::= TimeStampToken
```

The value of messageImprint field within **TimeStampToken** shall be a hash of the value of signature field within **SignerInfo** for the **signedData** being timestamped.

For further information and definition of TimeStampToken see subclause 10.4.

9.2 Complete Validation Data

An electronic signature with complete validation data is an Electronic Signature for which all the additional data required for validation (i.e. all certificates and revocation information) is available. Complete validation data (ES-C) build on the electronic signature Timestamp as defined above.

The minimum structure of a Complete validation data is:

- the Signature Timestamp Attribute, as defined in subclause 9.1.1;
- Complete Certificate Refs, as defined in subclause 9.2.1;
- Complete Revocation Refs, as defined in subclause 9.2.2.

The Complete validation data MAY also include the following additional information, forming a X-Long validation data, for use if later validation processes may not have access to this information:

- Complete Certificate Values, as defined in subclause 9.2.3;
- Complete Revocation Values, as defined in subclause 9.2.4.

The Complete validation data MAY also include one of the following additional attributes, forming a X-Timestamp validation data, to provide additional protection against later CA compromise and provide integrity of the validation data used:

- ES-C Timestamp, as defined in subclause 9.2.5; or
- Time-Stamped Certificates and CRLs references, as defined in subclause 9.2.6.

NOTE 1: As long as the CA's are trusted such that these keys cannot be compromised or the cryptography used broken, the ES-C provides long term proof of a valid electronic signature.

NOTE 2: The ES-C provides the following important property for long standing signatures; that is having been found once to be valid, shall continue to be so months or years later. Long after the validity period of the certificates have expired, or after the user key has been compromised.

9.2.1 Complete Certificate Refs Attribute Definition

The Complete Certificate Refs attribute is an unsigned attribute. It references the full set of CA certificates that have been used to validate a ES with Complete validation data up to (but not including) the signer's certificate. Only a single instance of this attribute shall occur with an electronic signature.

NOTE: The signer's certified is referenced in the signing certificate attribute (see subclause 8.1).

```
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 refs attribute value has the ASN.1 syntax CompleteCertificateRefs.

```
CompleteCertificateRefs ::=  SEQUENCE OF ETSICertID
```

ETSICertID is defined in subclause 8.8.2.

The **`IssuerSerial`** that shall be present in **ETSICertID**. The `certHash` shall match the hash of the certificate referenced.

NOTE: Copies of the certificate values may be held using the Certificate Values attribute defined in subclause 9.3.1.

9.2.2 Complete Revocation Refs Attribute Definition

The Complete Revocation Refs attribute is an unsigned attribute. Only a single instance of this attribute shall occur with an electronic signature. 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 ES with Complete validation data.

The following object identifier identifies the **CompleteRevocationRefs** 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 refs 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 **ETSICertID** in the **CompleteCertificateRefs** attribute. the second and subsequent **CrlOcspRef** fields shall be in the same order as the **ETSICertID** 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                  ETSIHash,
     crlIdentifier            CrlIdentifier OPTIONAL}


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


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

```
OcspResponsesID ::=  SEQUENCE {
    ocspIdentifier              OcspIdentifier,
    ocspRepHash                 ETSIHash    OPTIONAL
}


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

When creating an **crlValidatedID**, the **crlHash** is computed over the entire DER encoded CRL including the signature. The **crlIdentifier** would 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. The **crlListID** attribute is an unsigned attribute. 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 OSCP response using the issuer name and the time of issue of the OCSP response which shall correspond to the time "producedAt" contained in the issued OCSP response. Since it may be needed to make the difference between two OCSP responses received within the same second, then the hash of the response contained in the OcspResponsesID may be needed to solve the ambiguity.

NOTE: Copies of the CRL and OCSP responses values may be held using the Revocation Values attribute defined in subclause 9.3.2.

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

OtherRevRefType ::= OBJECT IDENTIFIER
```

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

9.3 Extended Validation Data

9.3.1 Certificate Values Attribute Definition

The Certificate Values attribute is an unsigned attribute. Only a single instance of this attribute shall occur with an electronic signature. It holds the values of certificates referenced in the CompleteCertificateRefs attribute.

NOTE: If an Attribute Certificate is used, it is not provided in this structure but shall be provided by the signer as a signer-attributes attribute (see subclause 12.3).

The following object identifier identifies the **CertificateValues** 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 subclause 10.1 (which is as defined in ITU-T Recommendation X.509 [1]).

9.3.2 Revocation Values Attribute Definition

The Revocation Values attribute is an unsigned attribute. Only a single instance of this attribute shall occur with an electronic signature. It holds the values of CRLs and OCSP referenced in the CompleteRevocationRefs attribute.

The following object identifier identifies the **CertificateValues** 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 }

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

OtherRevValType ::= OBJECT IDENTIFIER
```

The syntax and semantics of the other revocation values is 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 subclause 10.2 (which as defined in ITU-T Recommendation X.509 [1]).

BasicOCSPResponse is defined in subclause 10.3 (which as defined in RFC 2560 [8]).

9.3.3 ES-C Timestamp Attribute Definition

This attribute is used for the Type 1 X-Timestamped validation data. The ES-C Timestamp attribute is an unsigned attribute. It is timestamp of a hash of the electronic signature and the complete validation data (ES-C). It is a special purpose TimeStampToken Attribute which timestamps the ES-C. Several instances instance of this attribute may occur with an electronic signature from different TSAs.

The following object identifier identifies the ES-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 ES-C timestamp attribute value has the ASN.1 syntax **ESCTimeStampToken.**

```
ESCTimeStampToken ::= TimeStampToken
```

The value of messageImprint field within TimeStampToken shall be a hash of the concatenated values (without the type or length encoding for that value) of the following data objects as present in the ES with Complete validation data:

- signature field within SignerInfo;
- SignatureTimeStampToken attribute;
- CompleteCertificateRefs attribute;
- CompleteRevocationRefs attribute.

For further information and definition of the Time Stamp Token see subclause 10.4.

9.3.4 Time-Stamped Certificates and CRLs Attribute Definition

This attribute is used for the Type 2 X-Timestamp validation data. A **TimestampedCertsCRLsRef** attribute is an unsigned attribute. It is a list of referenced certificates and OCSP responses/CRLs which are been timestamped to protect against certain CA compromises. Its syntax is as follows:

The following object identifier identifies the **TimestampedCertsCRLsRef** 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
```

The value of messageImprint field within TimeStampToken shall be a hash of the concatenated values (without the type or length encoding for that value) of the following data objects as present in the ES with Complete validation data:

- CompleteCertificateRefs attribute;
- CompleteRevocationRefs attribute.

9.4 Archive Validation Data

Where an electronic signature is required to last for a very long time, and a the timestamp on an electronic signature is in danger of being invalidated due to algorithm weakness or limits in the validity period of the TSA certificate, then it may be required to timestamp the electronic signature several times. When this is required an archive timestamp attribute may be required. This timestamp may be repeatedly applied over a period of time.

9.4.1 Archive Timestamp Attribute Definition

The Archive Timestamp attribute is timestamp of the user data and the entire electronic signature. If the Certificate values and Revocation Values attributes are not present these attributes shall be added to the electronic signature prior to the timestamp. The Archive Timestamp attribute is an unsigned attribute. Several instances of this attribute may occur with on electronic signature both over time and from different TSAs.

The following object identifier identifies the Nested Archive Timestamp attribute:

```
id-aa-ets-archiveTimestamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 27}
```

Archive timestamp attribute values have the ASN.1 syntax **ArchiveTimeStampToken**

```
ArchiveTimeStampToken ::= TimeStampToken
```

The value of messageImprint field within TimeStampToken shall be a hash of the concatenated values (without the type or length encoding for that value) of the following data objects as present in the electronic signature:

- encapContentInfo eContent OCTET STRING;
- signedAttributes;
- signature field within SignerInfo;
- SignatureTimeStampToken attribute;
- CompleteCertificateRefs attribute;
- CompleteRevocationData attribute;
- CertificateValues attribute
  (If not already present this information shall be included in the ES-A);

- RevocationValues attribute
  (If not already present this information shall be included in the ES-A);
- ESCTimeStampToken attribute if present;
- TimestampedCertsCRLs attribute if present;
- any previous ArchiveTimeStampToken attributes.

For further information and definition of TimeStampToken see subclause 10.4.

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

10 Other standard data structures

This clause duplicates text and ASN.1 data structures that may be found in other standards, in particular the ITU-T Recommendation X.509 [1] and the IETF PKIX series of standards. All duplicated text and ASN.1 is provided in the present document for completeness and to avoid multiple reference to other documents with the aim of making it easier to read. Profiling the imported data structures to meet specific requirements of the present document provides value added. The copyright of the ITU-T Recommendation X.500 [15] remains with the ITU-T and the copyright of the PKIX text remain with the IETF.

10.1 Public-key Certificate Format

The X.509 v3 certificate basic syntax is as follows. For signature calculation, the certificate is encoded using the ASN.1 distinguished encoding rules (DER) (see ITU-T Recommendation X.690 [4]).

The syntax below is as defined in RFC 2459 [7] which is equivalent to that defined in ITU-T Recommendation X.509 [1].

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }
```

The field **tbsCertificate** contains the names of the subject and issuer, a public key associated with the subject, a validity period, and other associated information.

The **signatureAlgorithm** field contains the identifier for the cryptographic algorithm used by the CA to sign this certificate.

An **AlgorithmIdentifier** is defined by the following ASN.1 structure:

```
AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm               OBJECT IDENTIFIER,
     parameters              ANY DEFINED BY algorithm OPTIONAL  }
```

The **AlgorithmIdentifier** is used to identify a cryptographic algorithm. The OBJECT IDENTIFIER component identifies the algorithm (such as DSA with SHA-1). The contents of the optional parameters field will vary according to the algorithm identified. This field shall contain the same algorithm identifier as the signature field in the sequence **tbsCertificate**.

The **signatureValue** field contains a digital signature computed upon the ASN.1 DER encoded **tbsCertificate**. The ASN.1 DER encoded **tbsCertificate** is used as the input to the signature function. This signature value is then ASN.1 encoded as a BIT STRING and included in the Certificate's signature field.

By generating this signature, a CA certifies the validity of the information in the **tbsCertificate** field. In particular, the CA certifies the binding between the public key material and the subject of the certificate.

```
TBSCertificate  ::=  SEQUENCE  {
        version         [0]  EXPLICIT Version DEFAULT v1,
        serialNumber         CertificateSerialNumber,
        signature            AlgorithmIdentifier,
        issuer               Name,
        validity             Validity,
        subject              Name,
        subjectPublicKeyInfo SubjectPublicKeyInfo,
        issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                             -- If present, version shall be v2 or v3
        subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                             -- If present, version shall be v2 or v3
        extensions      [3]  EXPLICIT Extensions OPTIONAL
                             -- If present, version shall be v3
        }

   Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }

   CertificateSerialNumber  ::=  INTEGER

   Validity ::= SEQUENCE {
        notBefore      Time,
        notAfter       Time }


   Time ::= CHOICE {
        utcTime        UTCTime,
        generalTime    GeneralizedTime }
        UniqueIdentifier  ::=  BIT STRING

   SubjectPublicKeyInfo  ::=  SEQUENCE  {
        algorithm            AlgorithmIdentifier,
        subjectPublicKey     BIT STRING  }

   Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

   Extension  ::=  SEQUENCE  {
        extnID      OBJECT IDENTIFIER,
        critical    BOOLEAN DEFAULT FALSE,
        extnValue   OCTET STRING  }
```

The following syntax definitions associated with X.509 certificate extensions are used in the current document:

```
PolicyInformation ::= SEQUENCE {
        policyIdentifier   CertPolicyId,
        policyQualifiers   SEQUENCE SIZE (1..MAX) OF
                                PolicyQualifierInfo OPTIONAL }

   CertPolicyId ::= OBJECT IDENTIFIER

   PolicyQualifierInfo ::= SEQUENCE {
        policyQualifierId  PolicyQualifierId,
        qualifier          ANY DEFINED BY policyQualifierId }

PolicyQualifierId ::=
        OBJECT IDENTIFIER
```

10.2 Certificate Revocation List Format

The X.509 v2 CRL syntax is as follows. For signature calculation, the data that is to be signed is ASN.1 DER (see ITU-T Recommendation X.690 [4]) encoded. The syntax below is as defined in RFC 2459 [7] which is equivalent to that defined in ITU-T Recommendation X.509 [1].

```
   CertificateList  ::=  SEQUENCE  {
        tbsCertList          TBSCertList,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING  }

   TBSCertList  ::=  SEQUENCE  {
        version                 Version OPTIONAL,
                                     -- if present, shall be v2
        signature               AlgorithmIdentifier,
        issuer                  Name,
        thisUpdate              Time,
```

```
nextUpdate              Time OPTIONAL,
revokedCertificates     SEQUENCE OF SEQUENCE  {
     userCertificate         CertificateSerialNumber,
     revocationDate          Time,
     crlEntryExtensions      Extensions OPTIONAL
                                   -- if present, shall be v2
                           }  OPTIONAL,
crlExtensions           [0]  EXPLICIT Extensions OPTIONAL
                                   -- if present, shall be v2
                           }
```

The **CertificateList** is a SEQUENCE of three required fields.

The first field in the sequence is the **tbsCertList**. This field is itself a sequence containing the name of the issuer, issue date, issue date of the next list, the list of revoked certificates, and optional CRL extensions. Further, each entry on the revoked certificate list is defined by a sequence of user certificate serial number, revocation date, and optional CRL entry extensions.

The **signatureAlgorithm** field contains the algorithm identifier for the algorithm used by the CA to sign the **CertificateList**. The field is of type **AlgorithmIdentifier**, which was defined above.

This field shall contain the same algorithm identifier as the signature field in the sequence **tbsCertList**.

The **signatureValue** field contains a digital signature computed upon the ASN.1 DER encoded **tbsCertList**. The ASN.1 DER encoded **tbsCertList** is used as the input to the signature function. This signature value is then ASN.1 encoded as a BIT STRING and included in the CRL's **signatureValue** field.

The certificate list to be signed, or **TBSCertList**, is a SEQUENCE of required and optional fields. The required fields identify the CRL issuer, the algorithm used to sign the CRL, the date and time the CRL was issued, and the date and time by which the CA will issue the next CRL.

Optional fields include lists of revoked certificates and CRL extensions. The revoked certificate list is optional to support the case where a CA has not revoked any unexpired certificates that it has issued. The profile requires conforming CAs to use the CRL extension **cRLNumber** in all CRLs issued.

The optional field **Version** describes the version of the encoded CRL. When extensions are used, as required by this profile, this field shall be present and shall specify version 2 (the integer value is 1).

The Signature field contains the algorithm identifier for the algorithm used to sign the CRL. This field shall contain the same algorithm identifier as the **signatureAlgorithm** field in the sequence **CertificateList**.

The issuer name identifies the entity who has signed and issued the CRL. The issuer identity is carried in the issuer name field. Alternative name forms may also appear in the **issuerAltName** extension. The issuer name field shall contain an X.500 [15]] distinguished name (DN). The issuer name field is defined as the X.501 [16] type Name, and shall follow the encoding rules for the issuer name field in the certificate.

The field **thisUpdate** indicates the issue date of this CRL. **ThisUpdate** may be encoded as **UTCTime** or **GeneralizedTime**.

CAs conforming to this profile that issue CRLs shall encode **thisUpdate** as **UTCTime** for dates through the year 2049. CAs conforming to this profile that issue CRLs shall encode **thisUpdate** as **GeneralizedTime** for dates in the year 2050 or later.

The field **nextUpdate** indicates the date by which the next CRL will be issued. The next CRL could be issued before the indicated date, but it will not be issued any later than the indicated date. CAs SHOULD issue CRLs with a **nextUpdate** time equal to or later than all previous CRLs. **nextUpdate** may be encoded as **UTCTime** or **GeneralizedTime**.

This profile requires inclusion of **nextUpdate** in all CRLs issued by conforming CAs. Note that the ASN.1 syntax of **TBSCertList** describes this field as OPTIONAL, which is consistent with the ASN.1 structure defined in ITU-T Recommendation X.509 [1]. The behaviour of clients processing CRLs which omit **nextUpdate** is not specified by this profile.

Revoked Certificates

Revoked certificates are listed. The revoked certificates are named by their serial numbers. Certificates revoked by the CA are uniquely identified by the certificate serial number. The date on which the revocation occurred is specified.

Extensions

This field may only appear if the version is 2. If present, this field is a SEQUENCE of one or more CRL extensions.

The extensions defined by ANSI X9 and ISO/IEC/ITU for X.509 v2 CRLs provide methods for associating additional attributes with CRLs. The X.509 v2 CRL format also allows communities to define private extensions to carry information unique to those communities. Each extension in a CRL may be designated as critical or non-critical. A CRL verification shall fail if it encounters a critical extension which it does not know how to process. However, an unrecognized non-critical extension may be ignored. The following subclauses present those extensions used within Internet CRLs. Communities may elect to include extensions in CRLs which are not defined in this specification. However, caution should be exercised in adopting any critical extensions in CRLs which might be used in a general context.

The authority key identifier extension provides a means of identifying the public key corresponding to the private key used to sign a CRL. The identification can be based on either the key identifier (the subject key identifier in the CRL signer's certificate) or on the issuer name and serial number. This extension is especially useful where an issuer has more than one signing key, either due to multiple concurrent key pairs or due to changeover.

Conforming CAs shall use the key identifier method, and shall include this extension in all CRLs issued.

The issuer alternative names extension allows additional identities to be associated with the issuer of the CRL. Defined options include an RFC822 name (electronic mail address), a DNS name, an IP address, and a URI or URL. Multiple instances of a name and multiple name forms may be included. Whenever such identities are used, the issuer alternative name extension shall be used. The **issuerAltName** extension SHOULD NOT be marked critical.

The CRL number is a non-critical CRL extension which conveys a monotonically increasing sequence number for each CRL issued by a CA. This extension allows users to easily determine when a particular CRL supersedes another CRL. CAs conforming to this profile shall include this extension in all CRLs.

```
    id-ce-cRLNumber OBJECT IDENTIFIER ::= { joint-iso-ccitt(2) ds(5) 29 20 }
The syntax of cRLNumber is CRLNumber:

    CRLNumber ::= INTEGER (0..MAX)
```

Delta CRL Indicator

The delta CRL indicator is a critical CRL extension that identifies a delta-CRL. The use of delta-CRLs can significantly improve processing time for applications which store revocation information in a format other than the CRL structure. This allows changes to be added to the local database while ignoring unchanged information that is already in the local database.

When a delta-CRL is issued, the CAs shall also issue a complete CRL. The value of BaseCRLNumber identifies the CRL number of the base CRL that was used as the starting point in the generation of this delta-CRL. The delta-CRL contains the changes between the base CRL and the current CRL issued along with the delta-CRL. It is the decision of a CA as to whether to provide delta-CRLs. Again, a delta-CRL shall not be issued without a corresponding complete CRL. The value of CRLNumber for both the delta-CRL and the corresponding complete CRL shall be identical.

The following syntax definitions associated with X.509 certificate extensions are used in the current document:

```
CRLReason ::= ENUMERATED {
        unspecified             (0),
        keyCompromise           (1),
        cACompromise            (2),
        affiliationChanged      (3),
        superseded              (4),
        cessationOfOperation    (5),
        certificateHold         (6),
        removeFromCRL           (8) }
```

10.4 OCSP Response Format

The format of an OCSP token is defined by the structure "**BasicOCSPResponse**" imported from OCSP (RFC 2560 [8]):

```
   BasicOCSPResponse       ::= SEQUENCE {
      tbsResponseData      ResponseData,
      signatureAlgorithm   AlgorithmIdentifier,
      signature            BIT STRING,
      certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL
      }
```

The value for signature SHALL be computed on the hash of the DER (ITU-T Recommendation X.690 [4]) encoding **ResponseData**.

```
   ResponseData ::= SEQUENCE {
      version              [0] EXPLICIT Version DEFAULT v1,
      responderID              ResponderID,
      producedAt               GeneralizedTime,
      responses                SEQUENCE OF SingleResponse,
      responseExtensions   [1] EXPLICIT Extensions OPTIONAL }

   ResponderID ::= CHOICE {
      byName               [1] Name,
      byKey                [2] KeyHash }
```

The field "ResponderID" is there to indicate the ordering when several "SingleResponse" are present.

```
   KeyHash ::= OCTET STRING -- SHA-1 hash of responder's public key
   -- (excluding the tag and length fields)

   SingleResponse ::= SEQUENCE {
      certID                       CertID,
      certStatus                   CertStatus,
      thisUpdate                   GeneralizedTime,
      nextUpdate         [0]       EXPLICIT GeneralizedTime OPTIONAL,
      singleExtensions   [1]       EXPLICIT Extensions OPTIONAL }

   CertStatus ::= CHOICE {
       good        [0]     IMPLICIT NULL,
       revoked     [1]     IMPLICIT RevokedInfo,
       unknown     [2]     IMPLICIT UnknownInfo }

RevokedInfo ::= SEQUENCE {
       revocationTime              GeneralizedTime,
       revocationReason    [0]     EXPLICIT CRLReason OPTIONAL }

   UnknownInfo ::= NULL -- this can be replaced with an enumeration

CertID          ::=     SEQUENCE {
       hashAlgorithm       AlgorithmIdentifier,
       issuerNameHash      OCTET STRING, -- Hash of Issuer's DN
       issuerKeyHash       OCTET STRING, -- Hash of Issuers public key
       serialNumber        CertificateSerialNumber }
```

A valid OCSP token can only contain the value "good", otherwise it would be unuseful. Therefore the cases [1] and [2] will never appear.

The **thisUpdate** and **nextUpdate** fields define a recommended validity interval. This interval corresponds to the {**thisUpdate**, **nextUpdate**} interval in CRLs. Responses whose **nextUpdate** value is earlier than the local system time value SHOULD be considered unreliable.

Responses whose **thisUpdate** time is later than the local system time SHOULD be considered unreliable. Responses where the **nextUpdate** value is not set are equivalent to a CRL with no time for **nextUpdate** (see subclause 3.4).

The **producedAt** time is the time at which this response was signed.

10.5 Timestamping Token Format

The **timeStampToken** is defined in http://www.ietf.org/internet-drafts/draft-ietf-pkix-time-stamp-05.txt. This document is not yet stable and the reader shall consult the latest version or the RFC, when edited. A timestamp token attribute shall have a single attribute value.

A TimeStampToken is as follows. It is encapsulated as a SignedData construct [clause 8.4] in the EncapsulatedContentInfo field. The signed-data content type from [8.4] shall have ASN.1 type SignedData

The fields of type EncapsulatedContentInfo have the following meanings:

eContentType is an object identifier that uniquely specifies the content type. For a time stamping token it is defined as:

```
id-ct-TSTInfo  OBJECT IDENTIFIER ::= {id-ct 4}

with:

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

eContent is the content itself, carried as an octet string. The eContent content type shall have ASN.1 type TSTInfo. The time stamp token shall not contain any signatures other than the signature of the TSA. The certificate identifier of the TSA certificate shall be included as a signed attribute.

```
TSTInfo ::= SEQUENCE  {
     version                      Integer  { v1(1) },
     policy                       PolicyInformation,
     messageImprint               MessageImprint,
       -- MUST have the same value as the similar field in
       -- TimeStampReq
     serialNumber                 Integer,
     genTime                      GeneralizedTime,
     accuracy                     [0] Accuracy             OPTIONAL,
     nonce                        [1] Integer              OPTIONAL,
       -- MUST be present if the similar field was present
       -- in TimeStampReq. In that case it must have the same value.
     tsa                          [2] GeneralName          OPTIONAL,
     extensions                   [3] EXPLICIT Extensions  OPTIONAL
}
```

The **version** field (currently v1) describes the version of the Timestamp token.

Conforming timestamping servers shall be able to provide version 1.

Timestamp tokens. Among the optional fields, only the nonce field needs to be supported, if the similar field is present in TimeStampReq.

Conforming timestamping requesters shall be able to recognize version 1 Timestamp tokens with all the optional fields present, but are not mandated to understand the semantics of any extension, if present.

The **policy** field shall indicate the TSAs policy under which the response was produced. If a similar field was present in the TimeStampReq, then it shall have the same value, otherwise an error (badRequest) shall be returned. This policy MAY include the following types of information (although this list is certainly not exhaustive):

- The conditions under which the time stamp may be used.
- The availability of a time-stamp log, to allow later verification that a time-stamp token is authentic.

The **messageImprint** shall have the same value as the similar field in TimeStampReq, provided that the size of the hash value matches the expected size of the hash algorithm identified in hashAlgorithm.

The **serialNumber** field shall include a strictly monotonically increasing integer from one TimeStampToken to the next (e.g. 45, 236, 245, 1 023, ...). This guarantees that each token is unique and allows to compare the ordering of two time stamps from the same TSA. This is useful in particular when two times stamps from the same TSA bear the same time. This field also provides the way to build a unique identifier to reference the token. It should be noticed that the monotonic property shall remain valid even after a possible interruption (e.g. crash) of the service.

genTime is the time at which the timestamp has been created by the TSA. The ASN.1 GeneralizedTime syntax can include fraction-of-second details. Such syntax, without the restrictions from [RFC 2459 [7]] section 4.1.2.5.2, where GeneralizedTime is limited to represent time with one second, may to be used here. However, when there is no need to have a precision better than the second, then GeneralizedTime with a precision limited to one second should be used (as in RFC 2459 [7]).

The syntax is: YYYYMMDDhhmmss[.s...]Z

Example: 19990609001326.34352Z

ITU-T Recommendation X.690 | ISO/IEC 8825-1 [4] provides the restrictions for a DER-encoding.

The encoding shall terminate with a "Z". The decimal point element, if present, shall be the point option ".". The fractional-seconds elements, if present, shall omit all trailing 0's; if the elements correspond to 0, they shall be wholly omitted, and the decimal point element also shall be omitted.

Midnight (GMT) shall be represented in the form: "YYYYMMDD000000Z"where "YYYYMMDD" represents the day following the midnight in question.

Here are a few examples of valid representations:

"19920521000000Z"

"19920622123421Z"

"19920722132100.3Z"

accuracy represents the time deviation around the UTC time contained in GeneralizedTime.

Accuracy ::= CHOICE {

seconds [1] INTEGER,

millis [2] INTEGER (1..999),

micros [3] INTEGER (1..999)

}

By adding the accuracy value to the GeneralizedTime, an upper limit of the time at which the timestamp has been created by the TSA can be obtained. In the same way, by subtracting the accuracy to the GeneralizedTime, a lower limit of the time at which the timestamp has been created by the TSA can be obtained.

accuracy is expressed as an integer, either in seconds, milliseconds (between 1-999) or microseconds (1-999). When the accuracy field, which is optional, is missing, then, by default, an accuracy of one second is meant.

The **nonce** field shall be present if it was present in the TimeStampReq.

The purpose of the **tsa** field is to give an hint in identifying the name of the TSA. If present, it shall correspond to one of the subject names included in the certificate that is to be used to verify the token. However, the actual identification of the entity which signed the response will always occur through the use of the certificate identifier (ESSCertID Attribute or OtherCertID in subclause 8.8.2), which SHALL be present with the signed-data (see subclause 8.8).

extensions is a generic way to add additional information in the future. EXTENSIONS is defined in [RFC 2459]. However, the version 1 only supports non-critical extensions. This means that conforming requesters are not mandated to understand the semantics of any extension. Particular extension field types may be specified in standards or may be defined and registered by any organization or community.

10.6 Name and Attribute Formats

The syntax of the naming and other attributes used in the present document is as defined in RFC 2459 [7] which is based on ITU-T Recommedations X.400 [14], X.501 [16], X.509 [1] and X.520 [17], with amendment allow UTF-8 encoding of DirectoryString.

```
GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

GeneralName ::= CHOICE {
        otherName                       [0] AnotherName,
        rfc822Name                      [1] IA5String,
        dNSName                         [2] IA5String,
        x400Address                     [3] ORAddress,
        directoryName                   [4] Name,
        ediPartyName                    [5] EDIPartyName,
```

```
        uniformResourceIdentifier   [6] IA5String,
        iPAddress                   [7] OCTET STRING,
        registeredID                [8] OBJECT IDENTIFIER }

-- AnotherName replaces OTHER-NAME ::= TYPE-IDENTIFIER, as
-- TYPE-IDENTIFIER is not supported in the '88 ASN.1 syntax

AnotherName ::= SEQUENCE {

type-id    OBJECT IDENTIFIER,
     value      [0] EXPLICIT ANY DEFINED BY type-id }

EDIPartyName ::= SEQUENCE {
        nameAssigner        [0] DirectoryString {ub-name} OPTIONAL,
        partyName           [1] DirectoryString {ub-name} }


Attribute       ::=     SEQUENCE {
        type            AttributeType,
        values  SET OF AttributeValue
                -- at least one value is required -- }

AttributeType           ::=   OBJECT IDENTIFIER

AttributeValue          ::=   ANY

AttributeTypeAndValue           ::=     SEQUENCE {
        type    AttributeType,
        value   AttributeValue }

-- suggested naming attributes: Definition of the following
--  information object set may be augmented to meet local
--  requirements.  Note that deleting members of the set may
--  prevent interoperability with conforming implementations.
--  presented in pairs: the AttributeType followed by the
--  type definition for the corresponding AttributeValue

--Arc for standard naming attributes
id-at           OBJECT IDENTIFIER ::= {joint-iso-ccitt(2) ds(5) 4}

-- Attributes of type NameDirectoryString
id-at-name              AttributeType   ::=     {id-at 41}
id-at-surname           AttributeType   ::=     {id-at 4}
id-at-givenName         AttributeType   ::=     {id-at 42}
id-at-initials          AttributeType   ::=     {id-at 43}
id-at-generationQualifier       AttributeType   ::=     {id-at 44}

X520name        ::= CHOICE {
      teletexString         TeletexString (SIZE (1..ub-name)),
      printableString       PrintableString (SIZE (1..ub-name)),
      universalString       UniversalString (SIZE (1..ub-name)),
      utf8String            UTF8String (SIZE (1..ub-name)),
      bmpString             BMPString (SIZE(1..ub-name))   }

--

id-at-commonName        AttributeType   ::=     {id-at 3}

X520CommonName  ::=      CHOICE {
      teletexString         TeletexString (SIZE (1..ub-common-name)),
      printableString       PrintableString (SIZE (1..ub-common-name)),
      universalString       UniversalString (SIZE (1..ub-common-name)),
      utf8String            UTF8String (SIZE (1..ub-common-name)),
      bmpString             BMPString (SIZE(1..ub-common-name))   }

--

id-at-localityName      AttributeType   ::=     {id-at 7}

X520LocalityName ::= CHOICE {
      teletexString       TeletexString (SIZE (1..ub-locality-name)),
      printableString     PrintableString (SIZE (1..ub-locality-name)),
      universalString     UniversalString (SIZE (1..ub-locality-name)),
      utf8String          UTF8String (SIZE (1..ub-locality-name)),
      bmpString           BMPString (SIZE(1..ub-locality-name))   }

--

id-at-stateOrProvinceName       AttributeType   ::=     {id-at 8}

X520StateOrProvinceName         ::= CHOICE {
      teletexString       TeletexString (SIZE (1..ub-state-name)),
      printableString     PrintableString (SIZE (1..ub-state-name)),
```

```
      universalString     UniversalString (SIZE (1..ub-state-name)),
      utf8String          UTF8String (SIZE (1..ub-state-name)),
      bmpString           BMPString (SIZE(1..ub-state-name))   }

--

id-at-organizationName          AttributeType   ::=     {id-at 10}

X520OrganizationName ::= CHOICE {
  teletexString     TeletexString (SIZE (1..ub-organization-name)),
  printableString   PrintableString (SIZE (1..ub-organization-name)),
  universalString   UniversalString (SIZE (1..ub-organization-name)),
  utf8String        UTF8String (SIZE (1..ub-organization-name)),
  bmpString         BMPString (SIZE(1..ub-organization-name))   }

--

id-at-organizationalUnitName    AttributeType   ::=     {id-at 11}

X520OrganizationalUnitName ::= CHOICE {
 teletexString    TeletexString (SIZE (1..ub-organizational-unit-name)),
 printableString        PrintableString
                      (SIZE (1..ub-organizational-unit-name)),
 universalString        UniversalString
                      (SIZE (1..ub-organizational-unit-name)),
 utf8String       UTF8String (SIZE (1..ub-organizational-unit-name)),
 bmpString        BMPString (SIZE(1..ub-organizational-unit-name))   }

--

id-at-title     AttributeType   ::=     {id-at 12}

X520Title ::=   CHOICE {
      teletexString         TeletexString (SIZE (1..ub-title)),
      printableString       PrintableString (SIZE (1..ub-title)),
      universalString       UniversalString (SIZE (1..ub-title)),
      utf8String            UTF8String (SIZE (1..ub-title)),
      bmpString             BMPString (SIZE(1..ub-title))   }

--

id-at-dnQualifier       AttributeType   ::=     {id-at 46}
X520dnQualifier ::=     PrintableString

id-at-countryName       AttributeType   ::=     {id-at 6}
X520countryName ::=     PrintableString (SIZE (2)) -- IS 3166 codes

-- naming data types --

Name            ::=   CHOICE { -- only one possibility for now --
                                rdnSequence  RDNSequence }

RDNSequence     ::=   SEQUENCE OF RelativeDistinguishedName

DistinguishedName       ::=   RDNSequence

RelativeDistinguishedName  ::=
                    SET SIZE (1 .. MAX) OF AttributeTypeAndValue

-- Directory string type --

DirectoryString ::= CHOICE {
      teletexString             TeletexString (SIZE (1..MAX)),
      printableString           PrintableString (SIZE (1..MAX)),
      universalString           UniversalString (SIZE (1..MAX)),
      utf8String              UTF8String (SIZE (1..MAX)),
      bmpString               BMPString (SIZE(1..MAX))   }
-- x400 address syntax starts here
--      OR Names

ORAddress ::= SEQUENCE {
   built-in-standard-attributes BuiltInStandardAttributes,
   built-in-domain-defined-attributes
                        BuiltInDomainDefinedAttributes OPTIONAL,
   -- see also teletex-domain-defined-attributes
   extension-attributes ExtensionAttributes OPTIONAL }
--      The OR-address is semantically absent from the OR-name if the
--      built-in-standard-attribute sequence is empty and the
--      built-in-domain-defined-attributes and extension-attributes are
--      both omitted.

--      Built-in Standard Attributes
```

```
BuiltInStandardAttributes ::= SEQUENCE {
   country-name CountryName OPTIONAL,
   administration-domain-name AdministrationDomainName OPTIONAL,
   network-address      [0] NetworkAddress OPTIONAL,
   -- see also extended-network-address
   terminal-identifier  [1] TerminalIdentifier OPTIONAL,
   private-domain-name  [2] PrivateDomainName OPTIONAL,
   organization-name    [3] OrganizationName OPTIONAL,
   -- see also teletex-organization-name
   numeric-user-identifier      [4] NumericUserIdentifier OPTIONAL,
   personal-name        [5] PersonalName OPTIONAL,
   -- see also teletex-personal-name
   organizational-unit-names    [6] OrganizationalUnitNames OPTIONAL
   -- see also teletex-organizational-unit-names -- }

CountryName ::= [APPLICATION 1] CHOICE {
   x121-dcc-code NumericString
               (SIZE (ub-country-name-numeric-length)),
   iso-3166-alpha2-code PrintableString
               (SIZE (ub-country-name-alpha-length)) }

AdministrationDomainName ::= [APPLICATION 2] CHOICE {
   numeric NumericString (SIZE (0..ub-domain-name-length)),
   printable PrintableString (SIZE (0..ub-domain-name-length)) }

NetworkAddress ::= X121Address  -- see also extended-network-address

X121Address ::= NumericString (SIZE (1..ub-x121-address-length))

TerminalIdentifier ::= PrintableString (SIZE (1..ub-terminal-id-length))

PrivateDomainName ::= CHOICE {
   numeric NumericString (SIZE (1..ub-domain-name-length)),
   printable PrintableString (SIZE (1..ub-domain-name-length)) }

OrganizationName ::= PrintableString
                           (SIZE (1..ub-organization-name-length))
-- see also teletex-organization-name

NumericUserIdentifier ::= NumericString
                           (SIZE (1..ub-numeric-user-id-length))

PersonalName ::= SET {
   surname [0] PrintableString (SIZE (1..ub-surname-length)),
   given-name [1] PrintableString
                       (SIZE (1..ub-given-name-length)) OPTIONAL,
   initials [2] PrintableString (SIZE (1..ub-initials-length)) OPTIONAL,
   generation-qualifier [3] PrintableString
               (SIZE (1..ub-generation-qualifier-length)) OPTIONAL }
-- see also teletex-personal-name

OrganizationalUnitNames ::= SEQUENCE SIZE (1..ub-organizational-units)
                                       OF OrganizationalUnitName
-- see also teletex-organizational-unit-names

OrganizationalUnitName ::= PrintableString (SIZE
                       (1..ub-organizational-unit-name-length))

--      Built-in Domain-defined Attributes

BuiltInDomainDefinedAttributes ::= SEQUENCE SIZE
                                (1..ub-domain-defined-attributes) OF
                                BuiltInDomainDefinedAttribute

BuiltInDomainDefinedAttribute ::= SEQUENCE {
   type PrintableString (SIZE
                       (1..ub-domain-defined-attribute-type-length)),
   value PrintableString (SIZE
                       (1..ub-domain-defined-attribute-value-length))}


--      Extension Attributes

ExtensionAttributes ::= SET SIZE (1..ub-extension-attributes) OF
                        ExtensionAttribute

ExtensionAttribute ::=  SEQUENCE {
   extension-attribute-type [0] INTEGER (0..ub-extension-attributes),
   extension-attribute-value [1]
                        ANY DEFINED BY extension-attribute-type }


-- Extension types and attribute values
```

```
--

common-name INTEGER ::= 1

CommonName ::= PrintableString (SIZE (1..ub-common-name-length))

teletex-common-name INTEGER ::= 2

TeletexCommonName ::= TeletexString (SIZE (1..ub-common-name-length))

teletex-organization-name INTEGER ::= 3

TeletexOrganizationName ::=
               TeletexString (SIZE (1..ub-organization-name-length))

teletex-personal-name INTEGER ::= 4

TeletexPersonalName ::= SET {
   surname [0] TeletexString (SIZE (1..ub-surname-length)),
   given-name [1] TeletexString
               (SIZE (1..ub-given-name-length)) OPTIONAL,
   initials [2] TeletexString (SIZE (1..ub-initials-length)) OPTIONAL,
   generation-qualifier [3] TeletexString (SIZE
               (1..ub-generation-qualifier-length)) OPTIONAL }

teletex-organizational-unit-names INTEGER ::= 5

TeletexOrganizationalUnitNames ::= SEQUENCE SIZE
        (1..ub-organizational-units) OF TeletexOrganizationalUnitName

TeletexOrganizationalUnitName ::= TeletexString
                        (SIZE (1..ub-organizational-unit-name-length))

pds-name INTEGER ::= 7

PDSName ::= PrintableString (SIZE (1..ub-pds-name-length))

physical-delivery-country-name INTEGER ::= 8

PhysicalDeliveryCountryName ::= CHOICE {
   x121-dcc-code NumericString (SIZE (ub-country-name-numeric-length)),
   iso-3166-alpha2-code PrintableString
                        (SIZE (ub-country-name-alpha-length)) }

postal-code INTEGER ::= 9

PostalCode ::= CHOICE {
   numeric-code NumericString (SIZE (1..ub-postal-code-length)),
   printable-code PrintableString (SIZE (1..ub-postal-code-length)) }

physical-delivery-office-name INTEGER ::= 10

PhysicalDeliveryOfficeName ::= PDSParameter

physical-delivery-office-number INTEGER ::= 11

PhysicalDeliveryOfficeNumber ::= PDSParameter

extension-OR-address-components INTEGER ::= 12

ExtensionORAddressComponents ::= PDSParameter

physical-delivery-personal-name INTEGER ::= 13

PhysicalDeliveryPersonalName ::= PDSParameter

physical-delivery-organization-name INTEGER ::= 14

PhysicalDeliveryOrganizationName ::= PDSParameter

extension-physical-delivery-address-components INTEGER ::= 15

ExtensionPhysicalDeliveryAddressComponents ::= PDSParameter

unformatted-postal-address INTEGER ::= 16

UnformattedPostalAddress ::= SET {
   printable-address SEQUENCE SIZE (1..ub-pds-physical-address-lines) OF
           PrintableString (SIZE (1..ub-pds-parameter-length)) OPTIONAL,
   teletex-string TeletexString
         (SIZE (1..ub-unformatted-address-length)) OPTIONAL }

street-address INTEGER ::= 17
```

```
StreetAddress ::= PDSParameter

post-office-box-address INTEGER ::= 18

PostOfficeBoxAddress ::= PDSParameter

poste-restante-address INTEGER ::= 19

PosteRestanteAddress ::= PDSParameter

unique-postal-name INTEGER ::= 20

UniquePostalName ::= PDSParameter

local-postal-attributes INTEGER ::= 21

LocalPostalAttributes ::= PDSParameter

PDSParameter ::= SET {
   printable-string PrintableString
                (SIZE(1..ub-pds-parameter-length)) OPTIONAL,
   teletex-string TeletexString
                (SIZE(1..ub-pds-parameter-length)) OPTIONAL }

extended-network-address INTEGER ::= 22

ExtendedNetworkAddress ::= CHOICE {
   e163-4-address SEQUENCE {
        number [0] NumericString (SIZE (1..ub-e163-4-number-length)),
        sub-address [1] NumericString
                (SIZE (1..ub-e163-4-sub-address-length)) OPTIONAL },
   psap-address [0] PresentationAddress }

PresentationAddress ::= SEQUENCE {
        pSelector       [0] EXPLICIT OCTET STRING OPTIONAL,
        sSelector       [1] EXPLICIT OCTET STRING OPTIONAL,
        tSelector       [2] EXPLICIT OCTET STRING OPTIONAL,
        nAddresses      [3] EXPLICIT SET SIZE (1..MAX) OF OCTET STRING }

terminal-type  INTEGER ::= 23

TerminalType ::= INTEGER {
   telex (3),
   teletex (4),
   g3-facsimile (5),
   g4-facsimile (6),
   ia5-terminal (7),
   videotex (8) } (0..ub-integer-options)

--      Extension Domain-defined Attributes

teletex-domain-defined-attributes INTEGER ::= 6

TeletexDomainDefinedAttributes ::= SEQUENCE SIZE
   (1..ub-domain-defined-attributes) OF TeletexDomainDefinedAttribute

TeletexDomainDefinedAttribute ::= SEQUENCE {
        type TeletexString
               (SIZE (1..ub-domain-defined-attribute-type-length)),
        value TeletexString
               (SIZE (1..ub-domain-defined-attribute-value-length)) }

--  specifications of Upper Bounds shall be regarded as mandatory
--  from annex B of ITU-T X.411 Reference Definition of MTS Parameter
--  Upper Bounds

--      Upper Bounds
ub-name INTEGER ::=     32768
ub-common-name  INTEGER ::=     64
ub-locality-name        INTEGER ::=     128
ub-state-name   INTEGER ::=     128
ub-organization-name    INTEGER ::=     64
ub-organizational-unit-name     INTEGER ::=     64
ub-title        INTEGER ::=     64
ub-match        INTEGER ::=     128

ub-emailaddress-length INTEGER ::= 128

ub-common-name-length INTEGER ::= 64
ub-country-name-alpha-length INTEGER ::= 2
ub-country-name-numeric-length INTEGER ::= 3
ub-domain-defined-attributes INTEGER ::= 4
```

```
ub-domain-defined-attribute-type-length INTEGER ::= 8
ub-domain-defined-attribute-value-length INTEGER ::= 128
ub-domain-name-length INTEGER ::= 16
ub-extension-attributes INTEGER ::= 256
ub-e163-4-number-length INTEGER ::= 15
ub-e163-4-sub-address-length INTEGER ::= 40
ub-generation-qualifier-length INTEGER ::= 3
ub-given-name-length INTEGER ::= 16
ub-initials-length INTEGER ::= 5
ub-integer-options INTEGER ::= 256
ub-numeric-user-id-length INTEGER ::= 32
ub-organization-name-length INTEGER ::= 64
ub-organizational-unit-name-length INTEGER ::= 32
ub-organizational-units INTEGER ::= 4
ub-pds-name-length INTEGER ::= 16
ub-pds-parameter-length INTEGER ::= 30
ub-pds-physical-address-lines INTEGER ::= 6
ub-postal-code-length INTEGER ::= 16
ub-surname-length INTEGER ::= 40
ub-terminal-id-length INTEGER ::= 24
ub-unformatted-address-length INTEGER ::= 180
ub-x121-address-length INTEGER ::= 16

-- Note - upper bounds on string types, such as TeletexString, are
-- measured in characters.  Excepting PrintableString or IA5String, a
-- significantly greater number of octets will be required to hold
-- such a value.  As a minimum, 16 octets, or twice the specified upper
-- bound, whichever is the larger, should be allowed for TeletexString.
-- For UTF8String or UniversalString at least four times the upper
-- bound should be allowed.
```

11 Signature Policy Specification

The present document mandates that:

- an electronic signature shall be processed by the signer and verifier in accordance with the signature policy as identified by the signature policy attribute (see subclause 9.1);
- the signature policy shall be identifiable by an Object Identifier;
- there shall exist a specification of the signature policy;
- for a given signature policy there shall be one definitive form of the specification which has a unique binary encoding;
- a hash of the definitive specification, using an agreed algorithm, shall be provided by the signer and checked by the verifier (see subclause 9.1).

A signature policy specification includes general information about the policy, the validation policy rules and other signature policy information. Clause 6 describes the kind of information to be included in a signature policy..

The current document does not mandate the form of the signature policy specification. The signature policy may be specified either:

- in a free form document for human interpretation; or
- in a structured form using an agreed syntax and encoding.

The present document defines an ASN.1 based syntax that may be used to define a structured signature policy.

11.1 Overall ASN.1 Structure

The overall structure of a signature policy defined using ASN.1 is given in this clause. Use of this ASN.1 structure is optional.

This ASN.1 syntax is encoded using the distinguished encoding rules.

In this structure the policy information is preceded by an identifier for the hashing algorithm used to protect the signature policy and followed by the hash value which shall be re-calculated and checked whenever the policy is passed between the issuer and signer/verifier. The hash is calculated without the outer type and length fields.

```
SignaturePolicy ::= SEQUENCE {
    signPolicyHashAlg      AlgorithmIdentifier,
    signPolicyInfo         SignPolicyInfo,
    signPolicyHash         SignPolicyHash     OPTIONAL }

SignPolicyHash ::= OCTET STRING

SignPolicyInfo ::= SEQUENCE {
    signPolicyIdentifier            SignPolicyId,
    dateOfIssue                     GeneralizedTime,
    policyIssuerName                PolicyIssuerName,
    fieldOfApplication              FieldOfApplication,
    signatureValidationPolicy       SignatureValidationPolicy,
    signPolExtensions               SignPolExtensions       OPTIONAL
    }

SignPolicyId ::= OBJECT IDENTIFIER
```

The **policyIssuerName** field identifies the policy issuer in one or more of the general name forms.

```
PolicyIssuerName ::= GeneralNames
```

The **fieldofApplication** is a description of the expected application of this policy.

```
FieldOfApplication ::= DirectoryString
```

The signature validation policy rules are fully processable to allow the validation of electronic signatures issued under that signature policy. They are described in the rest of this clause.

11.2 Signature Validation Policy

The signature validation policy defines for the signer which data elements shall be present in the electronic signature he provides and for the verifier which data elements shall be present under that signature policy for an electronic signature to be potentially valid.

The signature validation policy is described as follows:

```
SignatureValidationPolicy ::= SEQUENCE {
    signingPeriod          SigningPeriod,
    commonRules            CommonRules,
    commitmentRules        CommitmentRules,
    signPolExtensions      SignPolExtensions        OPTIONAL
     }
```

The **signingPeriod** identifies the date and time before which the signature policy should not be used for creating signatures, and an optional date after which it should not be used for creating signatures.

```
SigningPeriod ::= SEQUENCE {
    notBefore   GeneralizedTime,
    notAfter    GeneralizedTime OPTIONAL }
```

11.3 Common Rules

The **CommonRules** define rules that are common to all commitment types. These rules are defined in terms of trust conditions for certificates, timestamps and attributes, along with any constraints on attributes that may be included in the electronic signature.

```
CommonRules  ::= SEQUENCE {
    signerAndVeriferRules          [0]  SignerAndVerifierRules     OPTIONAL,
    signingCertTrustCondition      [1]  SigningCertTrustCondition   OPTIONAL,
    timeStampTrustCondition        [2]  TimestampTrustCondition     OPTIONAL,
    attributeTrustCondition        [3]  AttributeTrustCondition     OPTIONAL,
    algorithmConstraintSet         [4]  AlgorithmConstraintSet      OPTIONAL,
    signPolExtensions              [5]  SignPolExtensions           OPTIONAL
    }
```

If a field is present in CommonRules then the equivalent field shall not be present in any of the CommitmentRules (see below). If any of the following fields are not present in CommonRules then it shall be present in each CommitmentRule:

- signerAndVeriferRules;
- signingCertTrustCondition;
- timeStampTrustCondition.

11.4 Commitment Rules

The **CommitmentRules** consists of the validation rules which apply to given commitment types:

```
CommitmentRules ::= SEQUENCE OF CommitmentRule
```

The **CommitmentRule** for given commitment types are defined in terms of trust conditions for certificates, timestamps and attributes, along with any constraints on attributes that may be included in the electronic signature.

```
CommitmentRule  ::= SEQUENCE {
    selCommitmentTypes                SelectedCommitmentTypes,
    signerAndVeriferRules        [0]  SignerAndVerifierRules      OPTIONAL,
    signingCertTrustCondition    [1]  SigningCertTrustCondition   OPTIONAL,
    timeStampTrustCondition      [2]  TimestampTrustCondition     OPTIONAL,
    attributeTrustCondition      [3]  AttributeTrustCondition     OPTIONAL,
    algorithmConstraintSet       [4]  AlgorithmConstraintSet      OPTIONAL,
    signPolExtensions            [5]  SignPolExtensions           OPTIONAL
    }

SelectedCommitmentTypes ::= SEQUENCE OF CHOICE {
    empty                        NULL,
    recognizedCommitmentType     CommitmentType }
```

If the SelectedCommitmentTypes indicates "*empty*" then this rule applied when a commitment type is not present (i.e.the type of commitment is indicated in the semantics of the message). Otherwise, the electronic signature shall contain a commitment type indication that shall fit one of the commitments types that are mentioned in CommitmentType.

A specific commitment type identifier shall not appear in more than one commitment rule.

```
CommitmentType ::= SEQUENCE {
    identifier          CommitmentTypeIdentifier,
    fieldOfApplication  [0] FieldOfApplication OPTIONAL,
    semantics           [1] DirectoryString OPTIONAL }
```

The **fieldOfApplication** and **semantics** fields define the specific use and meaning of the commitment within the overall field of application defined for the policy.

11.5 Signer and Verifier Rules

The SignerAndVerifierRules consists of signer rule and verification rules as defined below:

```
SignerAndVerifierRules ::= SEQUENCE {
    signerRules      SignerRules,
    verifierRules    VerifierRules }
```

11.5.1 Signer Rules

The signer rules identify:

- if the **eContent** is empty and the signature is calculated using a hash of signed data external to CMS structure;
- the CMS signed attributes that shall be provided by the signer under this policy;
- the CMS unsigned attribute that shall be provided by the signer under this policy;

- whether the certificate identifiers from the full certification path up to the trust point shall be provided by the signer in the **SigningCertificate** attribute;
- whether a signer's certificate, or all certificates in the certification path to the trust point shall be provided by the signer in the **certificates** field of **SignedData.**

```
SignerRules ::= SEQUENCE {
    externalSignedData         BOOLEAN  OPTIONAL,
            -- True if signed data is external to CMS structure
            -- False if signed data part of CMS structure
            -- not present if either allowed
    mandatedSignedAttr         CMSAttrs,    -- Mandated CMS signed attributes
    mandatedUnsignedAttr       CMSAttrs,    -- Mandated CMS unsigned attributed
    mandatedCertificateRef     [0] CertRefReq DEFAULT signerOnly,
                -- Mandated Certificate Reference
    mandatedCertificateInfo    [1] CertInfoReq DEFAULT none,
                -- Mandated Certificate Info
    signPolExtensions        [2] SignPolExtensions      OPTIONAL
                          }

CMSAttrs ::= SEQUENCE OF OBJECT IDENTIFIER
```

The **mandatedSignedAttr** field shall include the object identifier for all those signed attributes required by the present document as well as additional attributes required by this policy.

The **mandatedUnsignedAttr** field shall include the object identifier for all those unsigned attributes required by the present document as well as additional attributes required this policy. For example, if a signature timestamp (see subclause 1.1) is required *by the signer* the object identifier for this attribute shall be included.

The **mandatedCertificateRef** identifies whether just the signer's certificate, or all the full certificate path shall be provided by the signer.

```
CertRefReq ::= ENUMERATED {
                signerOnly (1),     -- Only reference to signer cert mandated
                fullPath (2)
                        -- References for full cert path up to a trust point required
                                    }
```

The **mandatedCertificateInfo** field identifies whether a signer's certificate, or all certificates in the certification path to the trust point shall be provided by the signer in the **certificates** field of **SignedData.**

```
CertInfoReq ::= ENUMERATED {
                none (0)    ,           -- No mandatory requirements
                signerOnly (1)  ,       -- Only reference to signer cert mandated
                fullPath (2)
                        -- References for full cert path up to a trust point mandated
        }
```

11.5.2 Verifier Rules

The verifier rules identify:

- The CMS unsigned attributes that shall be present under this policy and shall be added by the verifier if not added by the signer.

```
VerifierRules ::= SEQUENCE {
        mandatedUnsignedAttr    MandatedUnsignedAttr,
        signPolExtensions       SignPolExtensions       OPTIONAL
        }

MandatedUnsignedAttr ::=  CMSAttrs    -- Mandated CMS unsigned attributed
```

11.6 Certificate and Revocation Requirement

The **SigningCertTrustCondition**, **TimestampTrustCondition** and **AttributeTrustCondition (defined in subsequent subclauses)** make use of two ASN1 structures which are defined below: **CertificateTrustTrees** and **CertRevReq**.

11.6.1 Certificate Requirements

The **certificateTrustTrees** identifies a set of self signed certificates for the trust points used to start (or end) certificate path processing and the initial conditions for certificate path validation as defined RFC 2459 [7] section 6. This ASN1 structure is used to define policy for validating the signing certificate, the TSA's certificate and attribute certificates.

```
CertificateTrustTrees ::=   SEQUENCE OF CertificateTrustPoint

CertificateTrustPoint ::= SEQUENCE {
    trustpoint             Certificate,                       -- self-signed certificate
    pathLenConstraint   [0] PathLenConstraint   OPTIONAL,
    acceptablePolicySet [1] AcceptablePolicySet OPTIONAL,  -- If not present "any policy"
    nameConstraints     [2] NameConstraints     OPTIONAL,
    policyConstraints   [3] PolicyConstraints   OPTIONAL }
```

The **trustPoint** field gives the self signed certificate for the CA that is used as the trust point for the start of certificate path processing.

The **pathLenConstraint** field gives the maximum number of CA certificates that may be in a certification path following the **trustpoint**. A value of zero indicates that only the given **trustpoint** certificate and an end-entity certificate may be used. If present, the pathLenConstraint field shall be greater than or equal to zero. Where pathLenConstraint is not present, there is no limit to the allowed length of the certification path.

```
PathLenConstraint    ::=   INTEGER (0..MAX)
```

The **acceptablePolicySet** field identifies the initial set of certificate policies, any of which are acceptable under the signature policy.

```
AcceptablePolicySet ::= SEQUENCE OF CertPolicyId

CertPolicyId ::= OBJECT IDENTIFIER
```

The **nameConstraints** field indicates a name space within which all subject names in subsequent certificates in a certification path shall be located. Restrictions may apply to the subject distinguished name or subject alternative names. Restrictions apply only when the specified name form is present. If no name of the type is in the certificate, the certificate is acceptable.

Restrictions are defined in terms of permitted or excluded name subtrees. Any name matching a restriction in the **excludedSubtrees** field is invalid regardless of information appearing in the **permittedSubtrees**.

```
 NameConstraints ::= SEQUENCE {
          permittedSubtrees       [0]     GeneralSubtrees OPTIONAL,
          excludedSubtrees        [1]     GeneralSubtrees OPTIONAL }

      GeneralSubtrees ::= SEQUENCE SIZE (1..MAX) OF GeneralSubtree

      GeneralSubtree ::= SEQUENCE {
           base                    GeneralName,
           minimum         [0]     BaseDistance DEFAULT 0,
           maximum         [1]     BaseDistance OPTIONAL }

      BaseDistance ::= INTEGER (0..MAX)
```

The **policyConstraints** extension constrains path processing in two ways. It can be used to prohibit policy mapping or require that each certificate in a path contain an acceptable policy identifier.

The **policyConstraints** field, if present specifies requirement for explicit indication of the certificate policy and/or the constraints on policy mapping.

```
PolicyConstraints ::= SEQUENCE {
        requireExplicitPolicy           [0] SkipCerts OPTIONAL,
        inhibitPolicyMapping            [1] SkipCerts OPTIONAL }

SkipCerts ::= INTEGER (0..MAX)
```

If the **inhibitPolicyMapping** field is present, the value indicates the number of additional certificates that may appear in the path (including the trustpoint's self certificate) before policy mapping is no longer permitted. For example, a value of one indicates that policy mapping may be processed in certificates issued by the subject of this certificate, but not in additional certificates in the path.

If the **requireExplicitPolicy** field is present, subsequent certificates shall include an acceptable policy identifier. The value of **requireExplicitPolicy** indicates the number of additional certificates that may appear in the path (including the trustpoint's self certificate) before an explicit policy is required. An acceptable policy identifier is the identifier of a policy required by the user of the certification path or the identifier of a policy which has been declared equivalent through policy mapping.

11.6.2 Revocation Requirements

The **RevocRequirements** field specifies minimum requirements for revocation information, obtained through CRLs and/or OCSP responses, to be used in checking the revocation status of certificates. This ASN1 structure is used to define policy for validating the signing certificate, the TSA's certificate and attribute certificates.

```
CertRevReq ::= SEQUENCE {
    endCertRevReq   RevReq,
    caCerts         [0] RevReq
      }
```

Certificate revocation requirements are specified in terms of checks required on:

- **endCertRevReq**: end certificates (i.e. the signers certificate, the attribute certificate or the timestamping authority certificate);
- **caCerts**: CA certificates.

```
RevReq ::= SEQUENCE  {
    enuRevReq  EnuRevReq,
    exRevReq    SignPolExtensions OPTIONAL}

EnuRevReq  ::= ENUMERATED {
    clrCheck    (0), --Checks shall be made against current CRLs
            -- (or authority revocation lists)
    ocspCheck   (1), -- The revocation status shall be checked
            -- using the Online Certificate Status Protocol (RFC 2450)
    bothCheck   (2),    -- Both CRL and OCSP checks shall be carried out
    eitherCheck (3),    -- At least one of CRL or OCSP checks shall be carried out
    noCheck     (4),    -- no check is mandated
    other       (5)     -- Other mechanism as defined by signature poilicy extension
      }
```

Revocation requirements are specified in terms of:

- **clrCheck**: Checks shall be made against current CRLs (or authority revocation lists);
- **ocspCheck**: The revocation status shall be checked using the Online Certificate Status Protocol (RFC 2450 [21]);
- **bothCheck**: Both OCSP and CRL checks shall be carried out;
- **eitherCheck**: Either OCSP or CRL checks shall be carried out;
- **noCheck**: No check is mandated.

11.7 Signing Certificate Trust Conditions

The **SigningCertTrustCondition** field identifies trust conditions for certificate path processing used to validate the signing certificate.

```
SigningCertTrustCondition ::=   SEQUENCE {
     signerTrustTrees              CertificateTrustTrees,
     signerRevReq                  CertRevReq
}
```

11.8 TimeStamp Trust Conditions

The **TimeStampTrustCondition** field identifies trust conditions for certificate path processing used to authenticate the timstamping authority and constraints on the name of the timestamping authority. This applies to the timestamp that shall be present in every ES-T.

```
TimestampTrustCondition ::= SEQUENCE {
    ttsCertificateTrustTrees    [0]     CertificateTrustTrees    OPTIONAL,
    ttsRevReq                   [1]     CertRevReq              OPTIONAL,
    ttsNameConstraints          [2]     NameConstraints         OPTIONAL,
    cautionPeriod               [3]     DeltaTime               OPTIONAL,
    signatureTimestampDelay     [4]     DeltaTime               OPTIONAL }

DeltaTime ::= SEQUENCE {
    deltaSeconds    INTEGER,
    deltaMinutes    INTEGER,
    deltaHours      INTEGER,
    deltaDays       INTEGER }
```

If **ttsCertificateTrustTrees** is not present then the same rule as defined in **certificateTrustCondition** applies to certification of the timestamping authorities public key.

The **tstrRevReq** specifies minimum requirements for revocation information, obtained through CRLs and/or OCSP responses, to be used in checking the revocation status of the time stamp that shall be present in the ES-T.

If **ttsNameConstraints** is not present then there are no additional naming constraints on the trusted timestamping authority other than those implied by the **ttsCertificateTrustTrees**.

The **cautionPeriod** field specifies a caution period after the signing time that it is mandated the verifier shall wait to get high assurance of the validity of the signer's key and that any relevant revocation has been notified. The revocation status information forming the ES with Complete validation data shall not be collected and used to validate the electronic signature until after this caution period.

The **signatureTimestampDelay** field specifies a maximum acceptable time between the signing time and the time at which the signature timestamp, as used to form the ES Timestamped, is created for the verifier. If the signature timestamp is later that the time in the signing-time attribute by more than the value given in **signatureTimestampDelay**, the signature shall be considered invalid.

11.9 Attribute Trust Conditions

If the **attributeTrustCondition** field is not present then any certified attributes may not considered to be valid under this validation policy.

The **AttributeTrustCondition** field is defined as follows:

```
AttributeTrustCondition ::= SEQUENCE {
    attributeMandated           BOOLEAN,            -- Attribute shall be present
    howCertAttribute            HowCertAttribute,
    attrCertificateTrustTrees   [0] CertificateTrustTrees   OPTIONAL,
    attrRevReq                  [1] CertRevReq             OPTIONAL,
    attributeConstraints        [2] AttributeConstraints   OPTIONAL }
```

If **attributeMandated** is true then an attribute, certified within the following constraints, shall be present. If false, then the signature is still valid if no attribute is specified.

The **howCertAttribute** field specifies whether attributes uncertified attributes “claimed" by the signer, or certified in an attribute certificate or either using the signer attributes attribute defined in 8.12.3.

```
HowCertAttribute ::= ENUMERATED {
    claimedAttribute    (0),
    certifiedAttribtes  (1),
    either              (2) }
```

The **attrCertificateTrustTrees** specifies certificate path conditions for any attribute certificate. If not present the same rules apply as in **certificateTrustCondition**.

The **attrRevReq** specifies minimum requirements for revocation information, obtained through CRLs and/or OCSP responses, to be used in checking the revocation status of Attribute Certificates, if any are present.

If the **attributeConstraints** field is not present then there are no constraints on the attributes that may be validated under this policy. The **attributeConstraints** field is defined as follows:

```
AttributeConstraints ::= SEQUENCE {
    attributeTypeConstarints    [0] AttributeTypeConstraints  OPTIONAL,
    attributeValueConstarints   [1] AttributeValueConstraints OPTIONAL }
```

If present, the attributeTypeConstarints field specifies the attribute types which are considered valid under the signature policy. Any value for that attribute is considered valid.

```
AttributeTypeConstraints ::= SEQUENCE OF AttributeType
```

If present, the attributeTypeConstraints field specifies the specific attribute values which are considered valid under the signature policy.

```
AttributeValueConstraints ::= SEQUENCE OF AttributeTypeAndValue
```

11.10 Algorithm Constraints

The **algorithmConstrains** fields, if present, identifies the signing algorithms (hash, public key cryptography, combined hash and public key cryptography) that may be used for specific purposes and any minimum length. If this field is not present then the policy applies no constraints.

```
AlgorithmConstraintSet ::= SEQUENCE {   -- Algorithm constrains on:
signerAlgorithmConstraints  [0]     AlgorithmConstraints OPTIONAL, -- signer
eeCertAlgorithmConstraints  [1]     AlgorithmConstraints OPTIONAL, -- issuer of end entity certs.
caCertAlgorithmConstraints  [2]     AlgorithmConstraints OPTIONAL, -- issuer of CA certificates
aaCertAlgorithmConstraints  [3]     AlgorithmConstraints OPTIONAL, -- Attribute Authority
tsaCertAlgorithmConstraints [4]     AlgorithmConstraints OPTIONAL  -- TimeStamping Authority
             }

AlgorithmConstraints ::= SEQUENCE OF AlgAndLength

AlgAndLength ::= SEQUENCE {
    algID           OBJECT IDENTIFIER,
    minKeyLength    INTEGER     OPTIONAL, -- Minimum key length in bits
    other       SignPolExtensions OPTIONAL
         }
```

11.11 Signature Policy Extensions

Additional signature policy rules may be added to:

- the overall signature policy structure, as defined in subclause 11.1;
- the signature validation policy structure, as defined in subclause 11.2;
- the common rules, as defined in subclause 11.3;
- the commitment rules, as defined in subclause 11.4;
- the signer rules, as defined in subclause 11.5.1;
- the verifier rules, as defined in subclause 11.5.2;
- the revocation requirements in subclause 11.6.2;
- the algorithm constraints in subclause 11.10.

These extensions to the signature policy rules shall be defined using an ASN.1 syntax with an associated object identifier carried in the **SignPolExtn** as defined below:

```
SignPolExtensions ::= SEQUENCE OF SignPolExtn

SignPolExtn ::= SEQUENCE {
        extnID      OBJECT IDENTIFIER,
        extnValue   OCTET STRING  }
```

The **extnID** field shall contain the object identifier for the extension. The **extnValue** field shall contain the DER (see ITU-T Recommendation X.690 [4]) encoded value of the extension. The definition of an extension, as identified by **extnID** shall include a definition of the syntax and semantics of the extension.

12 Data protocols to interoperate with TSPs

12.1 Operational Protocols

The following protocols can be used by signers and verifiers to interoperate with Trusted Service Providers during the electronic signature creation and validation.

12.1.1 Certificate Retrieval

User certificates, CA certificate and cross-certificates can be retrieved from a repository using the Lightweight Directory Access Protocol as defined in RFC 1777 [6] and RFC 2559 [18], with the schema defined in RFC 2587 [19].

12.1.2 CRL Retrieval

Certificate revocation lists, including authority revocation lists and partial CRL variants, can be retrieved from a repository using the Lightweight Directory Access Protocol as defined in RFC 1777 [6] and RFC 2559 [18], with the schema defined in RFC 2587 [19].

12.1.3 OnLine Certificate Status

As an alternative to use of certificate revocation lists the status of certificate can be checked using the OnLine Certificate Status Protocol (OCSP) as defined in RFC 2560 [8].

12.1.4 Timestamping

The timestamping service can be accessed using the timestamping protocol currently defined in RFC (currently Internet Draft "draft-ietf-pkix-time-stamp-02.txt").

NOTE: This reference is to be replaced with the RFC number when published.

12.2 Management Protocols

Signers and verifiers can use the following management protocols to manage the use of certificates.

12.2.1 Certificate Request

Signers can request a public key certificate using the Certificate Request Message Format as defined in RFC 2511. This message format can be transported using a CMS signedData object as defined in RFC (currently Internet Draft "draft-ietf-pkix-cmc-05.txt").

NOTE: This reference is to be replaced with the RFC number when published.

Alternatively, the: "Internet Public Key Infrastructure Certificate Management Protocols". as defined in RFC 2510 [20] may be used.

12.2.2 Certificate Distribution to Signer

Certificates can be distributed to signers, transported using a CMS signedData object, as defined in RFC (currently, Internet Draft "draft-ietf-pkix-cmc-05.txt").

NOTE: This reference is to be replaced with the RFC number when published.

Alternatively, the: "Internet Public Key Infrastructure Certificate Management Protocols", as defined in RFC 2510 [20], may be used if this protocol is used in the request.

12.2.3 Request for Certificate Revocation

Signers and verifiers may request that a certificate is revoked using the revocation request and response messages defined in RFC 2510 [20].

13 Security considerations

13.1 Protection of Private Key

The security of the electronic signature mechanism defined in the present document depends on the privacy of the signer's private key. Implementations shall take steps to ensure that private keys cannot be compromised.

13.2 Choice of Algorithms

Implementers should be aware that cryptographic algorithms become weaker with time. As new cryptoanalysis techniques are developed and computing performance improves, the work factor to break a particular cryptographic algorithm will reduce. Therefore, cryptographic algorithm implementations should be modular allowing new algorithms to be readily inserted. That is, implementers should be prepared for the set of mandatory to implement algorithms to change over time.

14 Conformance Requirements

The present document only defines conformance requirements up to a ES with Complete validation data (ES-C). This means that none of the extended and archive forms of Electronic Signature (ES-X, ES-A) need to be implemented to get conformance to this standard.

The present document mandates support for elements of the signature policy.

14.1 Signer

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

- The general CMS syntax and content type as defined in RFC 2630 [9] (see subclauses 8.1 and 8.2).
- CMS SignedData as defined in RFC 2630 [9] with version set to 3 and at least one SignerInfo shall be present (see subclauses 8.3, 8.4, 8.5, 8.6).
- The following CMS Attributes as defined in RFC 2630 [9]:
  - ContentType; This shall always be present (see subclause 8.7.1);
  - MessageDigest; This shall always be present (see subclause 8.7.2);
  - SigningTime; This shall always be present (see subclause 8.7.3).

- The following ESS Attributes as defined in RFC 2634 [10]:
  - SigningCertificate: This shall be set as defined in subclauses 8.8.1 and 8.8.2.
- The following Attributes as defined in subclause 8.9:
  - SignaturePolicyIdentifier; This shall always be present.
- Public Key Certificates as defined in ITU-T Recommendation X.509 [1] and profiled in RFC 2459 [7] (see subclause 10.1).

14.2 Verifier

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

- Verification of the mandated components of an electronic signature, as defined in subclause 14.1.
- Signature Timestamp attribute, as defined in subclause 9.1.1.
- Complete Certificate Refs attribute, as defined in subclause 9.2.1.
- Complete Revocation Refs Attribute, as defined in subclause 9.2.2.
- Public Key Certificates, as defined in ITU-T Recommendation X.509 [1] and profiled in RFC 2459 [7] (see subclause 10.1)
- Either of:
  - Certificate Revocation Lists. as defined in ITU-T Recommendation X.509 [1] and profiled in RFC 2459 [7] (see subclause 10.2); or
  - On-line Certificate Status Protocol, as defined in RFC 2560 [8] (see subclause 10.3).

14.3 Signature Policy

Both signer and verifier systems shall be able to process an electronic signature in accordance with the specification of at least one signature policy, as identified by the signature policy attribute (see subclause 8.9.1).

Annex A (normative): ASN.1 Definitions

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

A.1 Definitions Using X.208 (1988) ASN.1 Syntax

NOTE: The ASN.1 module defined in clause A.1 has precedence over that defined in clause A.2 in the case of any conflict.

```
ETS-ElectronicSignature-88syntax { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-mod(0) 5}


DEFINITIONS EXPLICIT TAGS ::=
BEGIN
-- EXPORTS All -

IMPORTS

-- Crypographic Message Syntax (CMS): RFC 2630
    ContentInfo, ContentType, id-data, id-signedData, SignedData, EncapsulatedContentInfo,
    SignerInfo, id-contentType, id-messageDigest, MessageDigest, id-signingTime, SigningTime,
    id-countersignature, Countersignature
    FROM CryptographicMessageSyntax
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0) cms(1) }


-- ESS Defined attributes: RFC 2634 (Enhanced Security Services for S/MIME)
   id-aa-signingCertificate, SigningCertificate, IssuerSerial,
    id-aa-contentReference, ContentReference, id-aa-contentIdentifier, ContentIdentifier
    FROM ExtendedSecurityServices
     { iso(1) member-body(2) us(840) rsadsi(113549)
       pkcs(1) pkcs-9(9) smime(16) modules(0) ess(2) }


-- Internet X.509 Public Key Infrastructure - Certificate and CRL Profile: RFC 2459
    Certificate, AlgorithmIdentifier, CertificateList, Name, GeneralNames, GeneralName,
    DirectoryString,Attribute, AttributeTypeAndValue, AttributeType, AttributeValue,
     PolicyInformation, 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-88(1)}

-- X.509 '97 Authentication Framework
AttributeCertificate
    FROM AuthenticationFramework
    {joint-iso-ccitt ds(5) module(1) authenticationFramework(7) 3}
-- The imported AttributeCertificate is defined using the X.680 1997 ASN.1 Syntax,
-- an equivalent using the 88 ASN.1 syntax may be used.


-- OCSP 2560
BasicOCSPResponse, ResponderID
    FROM OCSP {-- OID not assigned -- }


-- Time Stamp Protocol Internet Draft
TimeStampToken
    FROM TSP {-- OID not assigned -- };


-- S/MIME Object Identifier arcs used in the present document
-- ==========================================================

-- S/MIME  OID arc used in the present document
-- id-smime OBJECT IDENTIFIER ::= { iso(1) member-body(2)
--             us(840) rsadsi(113549) pkcs(1) pkcs-9(9) 16 }
```

```
-- S/MIME Arcs
-- id-mod  OBJECT IDENTIFIER ::= { id-smime 0 }   -- modules
-- id-ct   OBJECT IDENTIFIER ::= { id-smime 1 }   -- content types
-- id-aa   OBJECT IDENTIFIER ::= { id-smime 2 }   -- attributes
-- id-spq  OBJECT IDENTIFIER ::= { id-smime 5 }   -- signature policy qualifier
-- id-cti  OBJECT IDENTIFIER ::= { id-smime 6 }   -- commitment type identifier


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

-- The allocation of OIDs to specific objects are given below with the associated
-- ASN.1 syntax definition

-- OID used referencing electronic signature mechanisms based on this standard
-- 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) }



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

-- Mandatory Electronic Signature Attributes

-- OtherSigningCertificate

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}

OtherHashValue ::= OCTET STRING

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


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

SigPolicyId ::= OBJECT IDENTIFIER

SigPolicyHash ::= ETSIHashAlgAndValue

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)) }


-- Optional Electronic Signature Attributes

-- 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 {
        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

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

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 X.509 : see section 10.3

-- 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

-- Validation Data

-- 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 ETSICertID


-- 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                  ETSIHash,
     crlIdentifier            CrlIdentifier OPTIONAL}

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

OcspListID ::=  SEQUENCE {
    ocspResponses        SEQUENCE OF OcspResponsesID}

OcspResponsesID ::=  SEQUENCE {
    ocspIdentifier             OcspIdentifier,
    ocspRepHash                ETSIHash    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

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 }

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

OtherRevValType ::= OBJECT IDENTIFIER


-- ES-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-archiveTimestamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 27}

ArchiveTimeStampToken ::= TimeStampToken


-- Signature Policy Specification
-- ==============================

SignaturePolicy ::= SEQUENCE {
    signPolicyHashAlg      AlgorithmIdentifier,
    signPolicyInfo         SignPolicyInfo,
    signPolicyHash         SignPolicyHash     OPTIONAL }

SignPolicyHash ::= OCTET STRING

SignPolicyInfo ::= SEQUENCE {
    signPolicyIdentifier            SignPolicyId,
    dateOfIssue                     GeneralizedTime,
    policyIssuerName                PolicyIssuerName,
    fieldOfApplication              FieldOfApplication,
    signatureValidationPolicy       SignatureValidationPolicy,
    signPolExtensions               SignPolExtensions       OPTIONAL
    }

SignPolicyId ::= OBJECT IDENTIFIER

PolicyIssuerName ::= GeneralNames

FieldOfApplication ::= DirectoryString

SignatureValidationPolicy ::= SEQUENCE {
    signingPeriod          SigningPeriod,
    commonRules            CommonRules,
    commitmentRules        CommitmentRules,
    signPolExtensions      SignPolExtensions        OPTIONAL
     }

SigningPeriod ::= SEQUENCE {
    notBefore   GeneralizedTime,
    notAfter    GeneralizedTime OPTIONAL }

CommonRules  ::= SEQUENCE {
    signerAndVeriferRules          [0]  SignerAndVerifierRules     OPTIONAL,
```

```
    signingCertTrustCondition      [1]  SigningCertTrustCondition   OPTIONAL,
    timeStampTrustCondition        [2]  TimestampTrustCondition     OPTIONAL,
    attributeTrustCondition        [3]  AttributeTrustCondition     OPTIONAL,
    algorithmConstraintSet         [4]  AlgorithmConstraintSet      OPTIONAL,
    signPolExtensions              [5]  SignPolExtensions           OPTIONAL
    }

CommitmentRules ::= SEQUENCE OF CommitmentRule

CommitmentRule  ::= SEQUENCE {
    selCommitmentTypes                  SelectedCommitmentTypes,
    signerAndVeriferRules          [0]  SignerAndVerifierRules      OPTIONAL,
    signingCertTrustCondition      [1]  SigningCertTrustCondition   OPTIONAL,
    timeStampTrustCondition        [2]  TimestampTrustCondition     OPTIONAL,
    attributeTrustCondition        [3]  AttributeTrustCondition     OPTIONAL,
    algorithmConstraintSet         [4]  AlgorithmConstraintSet      OPTIONAL,
    signPolExtensions              [5]  SignPolExtensions           OPTIONAL
    }


SelectedCommitmentTypes ::= SEQUENCE OF CHOICE {
    empty                       NULL,
    recognizedCommitmentType    CommitmentType }

CommitmentType ::= SEQUENCE {
    identifier          CommitmentTypeIdentifier,
    fieldOfApplication  [0] FieldOfApplication OPTIONAL,
    semantics           [1] DirectoryString OPTIONAL }

SignerAndVerifierRules ::= SEQUENCE {
    signerRules      SignerRules,
    verifierRules    VerifierRules }

SignerRules ::= SEQUENCE {
    externalSignedData         BOOLEAN  OPTIONAL,
            -- True if signed data is external to CMS structure
            -- False if signed data part of CMS structure
            -- not present if either allowed
    mandatedSignedAttr         CMSAttrs,    -- Mandated CMS signed attributes
    mandatedUnsignedAttr       CMSAttrs,    -- Mandated CMS unsigned attributed
    mandatedCertificateRef     [0] CertRefReq DEFAULT signerOnly,
                -- Mandated Certificate Reference
    mandatedCertificateInfo    [1] CertInfoReq DEFAULT none,
                -- Mandated Certificate Info
    signPolExtensions        [2] SignPolExtensions      OPTIONAL
                          }

CMSAttrs ::= SEQUENCE OF OBJECT IDENTIFIER

CertRefReq ::= ENUMERATED {
                signerOnly (1),     -- Only reference to signer cert mandated
                fullPath (2)
                        -- References for full cert path up to a trust point required
                                  }

CertInfoReq ::= ENUMERATED {
                none (0)    ,           -- No mandatory requirements
                signerOnly (1)  ,       -- Only reference to signer cert mandated
                fullPath (2)
                        -- References for full cert path up to a trust point mandated
        }

VerifierRules ::= SEQUENCE {
        mandatedUnsignedAttr    MandatedUnsignedAttr,
        signPolExtensions       SignPolExtensions       OPTIONAL
        }

MandatedUnsignedAttr ::=  CMSAttrs    -- Mandated CMS unsigned attributed


CertificateTrustTrees ::=   SEQUENCE OF CertificateTrustPoint

CertificateTrustPoint ::= SEQUENCE {
    trustpoint              Certificate,                    -- self-signed certificate
    pathLenConstraint   [0] PathLenConstraint   OPTIONAL,
    acceptablePolicySet [1] AcceptablePolicySet OPTIONAL,  -- If not present "any policy"
    nameConstraints     [2] NameConstraints     OPTIONAL,
    policyConstraints   [3] PolicyConstraints   OPTIONAL }

PathLenConstraint    ::=   INTEGER (0..MAX)

AcceptablePolicySet ::= SEQUENCE OF CertPolicyId
```

```
CertPolicyId ::= OBJECT IDENTIFIER

NameConstraints ::= SEQUENCE {
          permittedSubtrees       [0]     GeneralSubtrees OPTIONAL,
          excludedSubtrees        [1]     GeneralSubtrees OPTIONAL }

      GeneralSubtrees ::= SEQUENCE SIZE (1..MAX) OF GeneralSubtree

      GeneralSubtree ::= SEQUENCE {
           base                    GeneralName,
           minimum         [0]     BaseDistance DEFAULT 0,
           maximum         [1]     BaseDistance OPTIONAL }

      BaseDistance ::= INTEGER (0..MAX)

PolicyConstraints ::= SEQUENCE {
        requireExplicitPolicy           [0] SkipCerts OPTIONAL,
        inhibitPolicyMapping            [1] SkipCerts OPTIONAL }

SkipCerts ::= INTEGER (0..MAX)

CertRevReq ::= SEQUENCE {
    endCertRevReq   RevReq,
    caCerts         [0] RevReq
      }

RevReq ::= SEQUENCE  {
    enuRevReq  EnuRevReq,
    exRevReq    SignPolExtensions OPTIONAL}

EnuRevReq  ::= ENUMERATED {
    clrCheck    (0), --Checks shall be made against current CRLs
            -- (or authority revocation lists)
    ocspCheck   (1), -- The revocation status shall be checked
            -- using the Online Certificate Status Protocol (RFC 2450)
    bothCheck   (2),    -- Both CRL and OCSP checks shall be carried out
    eitherCheck (3),    -- At least one of CRL or OCSP checks shall be carried out
    noCheck     (4),    -- no check is mandated
    other       (5)     -- Other mechanism as defined by signature poilicy extension
      }

SigningCertTrustCondition ::=   SEQUENCE {
     signerTrustTrees              CertificateTrustTrees,
     signerRevReq                  CertRevReq
}

TimestampTrustCondition ::= SEQUENCE {
    ttsCertificateTrustTrees    [0]     CertificateTrustTrees    OPTIONAL,
    ttsRevReq                   [1]     CertRevReq              OPTIONAL,
    ttsNameConstraints          [2]     NameConstraints         OPTIONAL,
    cautionPeriod               [3]     DeltaTime               OPTIONAL,
    signatureTimestampDelay     [4]     DeltaTime               OPTIONAL }

DeltaTime ::= SEQUENCE {
    deltaSeconds    INTEGER,
    deltaMinutes    INTEGER,
    deltaHours      INTEGER,
    deltaDays       INTEGER }

AttributeTrustCondition ::= SEQUENCE {
    attributeMandated            BOOLEAN,              -- Attribute shall be present
    howCertAttribute             HowCertAttribute,
    attrCertificateTrustTrees   [0] CertificateTrustTrees   OPTIONAL,
    attrRevReq                  [1] CertRevReq             OPTIONAL,
    attributeConstraints        [2] AttributeConstraints   OPTIONAL }


HowCertAttribute ::= ENUMERATED {
    claimedAttribute    (0),
    certifiedAttribtes  (1),
    either              (2) }

AttributeConstraints ::= SEQUENCE {
    attributeTypeConstarints    [0] AttributeTypeConstraints  OPTIONAL,
    attributeValueConstarints   [1] AttributeValueConstraints OPTIONAL }


AttributeTypeConstraints ::= SEQUENCE OF AttributeType

AttributeValueConstraints ::= SEQUENCE OF AttributeTypeAndValue
```

```
AlgorithmConstraintSet ::= SEQUENCE {   -- Algorithm constrains on:
signerAlgorithmConstraints  [0]     AlgorithmConstraints OPTIONAL, -- signer
eeCertAlgorithmConstraints  [1]     AlgorithmConstraints OPTIONAL, -- issuer of end entity certs.
caCertAlgorithmConstraints  [2]     AlgorithmConstraints OPTIONAL, -- issuer of CA certificates
aaCertAlgorithmConstraints  [3]     AlgorithmConstraints OPTIONAL, -- Attribute Authority
tsaCertAlgorithmConstraints [4]     AlgorithmConstraints OPTIONAL  -- TimeStamping Authority
            }

AlgorithmConstraints ::= SEQUENCE OF AlgAndLength

AlgAndLength ::= SEQUENCE {
    algID           OBJECT IDENTIFIER,
    minKeyLength    INTEGER     OPTIONAL, -- Minimum key length in bits
    other       SignPolExtensions OPTIONAL
         }

SignPolExtensions ::= SEQUENCE OF SignPolExtn

SignPolExtn ::= SEQUENCE {
        extnID      OBJECT IDENTIFIER,
        extnValue   OCTET STRING  }

END -- ETS-ElectronicSignature-88syntax --
```

A.2 Definitions Using X.680 1997 ASN.1 Syntax

NOTE: The ASN.1 module defined in clause A.1 has precedence over that defined in clause A.2 in the case of any conflict.

```
ETS-ElectronicSignature-97Syntax { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-mod(0) 6}

DEFINITIONS EXPLICIT TAGS ::=
BEGIN
-- EXPORTS All -

IMPORTS

-- Crypographic Message Syntax (CMS): RFC 2630
    ContentInfo, ContentType, id-data, id-signedData, SignedData,
    EncapsulatedContentInfo, SignerInfo,
    id-contentType, id-messageDigest, MessageDigest, id-signingTime, SigningTime,
    id-countersignature, Countersignature
   FROM CryptographicMessageSyntax
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0) cms(1) }


-- ESS Defined attributes: RFC 2634 (Enhanced Security Services for S/MIME)
   id-aa-signingCertificate, SigningCertificate, IssuerSerial,
    id-aa-contentReference, ContentReference, id-aa-contentIdentifier, ContentIdentifier
    FROM ExtendedSecurityServices
    { iso(1) member-body(2) us(840) rsadsi(113549)
       pkcs(1) pkcs-9(9) smime(16) modules(0) ess(2) }


-- Internet X.509 Public Key Infrastructure - Certificate and CRL Profile: RFC 2459
    Certificate, AlgorithmIdentifier, CertificateList, Name, GeneralNames, GeneralName,
    DirectoryString, Attribute, AttributeTypeAndValue, AttributeType, AttributeValue,
    PolicyInformation
  FROM PKIX1Explicit93
    {iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-explicit-88(1)}

-- X.509 '97 Authentication Framework
AttributeCertificate
  FROM AuthenticationFramework
        {joint-iso-ccitt ds(5) module(1) authenticationFramework(7) 3}


-- OCSP 2560
BasicOCSPResponse, ResponderID
    FROM OCSP
--  { OID not assigned }


-- Time Stamp Protocol Internet Draft
```

```
TimeStampToken
    FROM TSP
-- { OID not assigned };


-- S/MIME Object Identifier arcs used in the present document
-- ==========================================================

-- S/MIME  OID arc used in the present document
-- id-smime OBJECT IDENTIFIER ::= { iso(1) member-body(2)
--             us(840) rsadsi(113549) pkcs(1) pkcs-9(9) 16 }

-- S/MIME Arcs
-- id-mod  OBJECT IDENTIFIER ::= { id-smime 0 }   -- modules
-- id-ct   OBJECT IDENTIFIER ::= { id-smime 1 }   -- content types
-- id-aa   OBJECT IDENTIFIER ::= { id-smime 2 }   -- attributes
-- id-spq  OBJECT IDENTIFIER ::= { id-smime 5 }   -- signature policy qualifier
-- id-cti  OBJECT IDENTIFIER ::= { id-smime 6 }   -- commitment type identifier


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


-- The allocation of OIDs to specific objects are given below with the associated
-- ASN.1 syntax definition

-- OID used referencing electronic signature mechanisms based on this standard
-- 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) }



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

-- Mandatory Electronic Signature Attributes

-- OtherSigningCertificate

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}

OtherHashValue ::= OCTET STRING

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


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

SigPolicyId ::= OBJECT IDENTIFIER

SigPolicyHash ::= ETSIHashAlgAndValue
```

```
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-sqt-unotice SIG-QUALIFIER-TYPE SPUserNotice }

pointerToSigPolSpec SIG-POLICY-QUALIFIER ::= {
      SIG-POLICY-QUALIFIER-ID id-sqt-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)) }


-- Optional Electronic Signature Attributes

-- 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


-- 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 X.509 : see section 10.3


-- 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


-- Validation Data

-- 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 ETSICertID


-- 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                  ETSIHash,
```

```
     crlIdentifier             CrlIdentifier OPTIONAL}

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

OcspListID ::=  SEQUENCE {
    ocspResponses        SEQUENCE OF OcspResponsesID}

OcspResponsesID ::=  SEQUENCE {
    ocspIdentifier             OcspIdentifier,
    ocspRepHash                ETSIHash    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 }

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 }


-- ES-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-archiveTimestamp OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 27}
```

```
ArchiveTimeStampToken ::= TimeStampToken


-- Signature Policy Specification
-- ==============================

SignaturePolicy ::= SEQUENCE {
    signPolicyHashAlg      AlgorithmIdentifier,
    signPolicyInfo         SignPolicyInfo,
    signPolicyHash         SignPolicyHash     OPTIONAL }

SignPolicyHash ::= OCTET STRING

SignPolicyInfo ::= SEQUENCE {
    signPolicyIdentifier            SignPolicyId,
    dateOfIssue                     GeneralizedTime,
    policyIssuerName                PolicyIssuerName,
    fieldOfApplication              FieldOfApplication,
    signatureValidationPolicy       SignatureValidationPolicy,
    signPolExtensions               SignPolExtensions       OPTIONAL
    }

SignPolicyId ::= OBJECT IDENTIFIER

PolicyIssuerName ::= GeneralNames

FieldOfApplication ::= DirectoryString

SignatureValidationPolicy ::= SEQUENCE {
    signingPeriod          SigningPeriod,
    commonRules            CommonRules,
    commitmentRules        CommitmentRules,
    signPolExtensions      SignPolExtensions        OPTIONAL
     }

SigningPeriod ::= SEQUENCE {
    notBefore   GeneralizedTime,
    notAfter    GeneralizedTime OPTIONAL }

CommonRules  ::= SEQUENCE {
    signerAndVeriferRules          [0]  SignerAndVerifierRules     OPTIONAL,
    signingCertTrustCondition      [1]  SigningCertTrustCondition   OPTIONAL,
    timeStampTrustCondition        [2]  TimestampTrustCondition     OPTIONAL,
    attributeTrustCondition        [3]  AttributeTrustCondition     OPTIONAL,
    algorithmConstraintSet         [4]  AlgorithmConstraintSet      OPTIONAL,
    signPolExtensions              [5]  SignPolExtensions           OPTIONAL
    }

CommitmentRules ::= SEQUENCE OF CommitmentRule

CommitmentRule  ::= SEQUENCE {
    selCommitmentTypes                  SelectedCommitmentTypes,
    signerAndVeriferRules          [0]  SignerAndVerifierRules      OPTIONAL,
    signingCertTrustCondition      [1]  SigningCertTrustCondition   OPTIONAL,
    timeStampTrustCondition        [2]  TimestampTrustCondition     OPTIONAL,
    attributeTrustCondition        [3]  AttributeTrustCondition     OPTIONAL,
    algorithmConstraintSet         [4]  AlgorithmConstraintSet      OPTIONAL,
    signPolExtensions              [5]  SignPolExtensions           OPTIONAL
    }

SelectedCommitmentTypes ::= SEQUENCE OF CHOICE {
    empty                       NULL,
    recognizedCommitmentType    CommitmentType }

CommitmentType ::= SEQUENCE {
    identifier          CommitmentTypeIdentifier,
    fieldOfApplication  [0] FieldOfApplication OPTIONAL,
    semantics           [1] DirectoryString OPTIONAL }

SignerAndVerifierRules ::= SEQUENCE {
    signerRules      SignerRules,
    verifierRules    VerifierRules }

SignerRules ::= SEQUENCE {
    externalSignedData         BOOLEAN  OPTIONAL,
            -- True if signed data is external to CMS structure
            -- False if signed data part of CMS structure
            -- not present if either allowed
    mandatedSignedAttr         CMSAttrs,    -- Mandated CMS signed attributes
    mandatedUnsignedAttr       CMSAttrs,    -- Mandated CMS unsigned attributed
    mandatedCertificateRef     [0] CertRefReq DEFAULT signerOnly,
                -- Mandated Certificate Reference
    mandatedCertificateInfo    [1] CertInfoReq DEFAULT none,
```

```
                -- Mandated Certificate Info
    signPolExtensions       [2] SignPolExtensions      OPTIONAL
                        }

CMSAttrs ::= SEQUENCE OF OBJECT IDENTIFIER

CertRefReq ::= ENUMERATED {
                signerOnly (1),     -- Only reference to signer cert mandated
                fullPath (2)
                        -- References for full cert path up to a trust point required
                                }

CertInfoReq ::= ENUMERATED {
                none (0)    ,           -- No mandatory requirements
                signerOnly (1)  ,       -- Only reference to signer cert mandated
                fullPath (2)
                        -- References for full cert path up to a trust point mandated
        }

VerifierRules ::= SEQUENCE {
        mandatedUnsignedAttr    MandatedUnsignedAttr,
        signPolExtensions       SignPolExtensions       OPTIONAL
        }

MandatedUnsignedAttr ::=  CMSAttrs    -- Mandated CMS unsigned attributed


CertificateTrustTrees ::=   SEQUENCE OF CertificateTrustPoint

CertificateTrustPoint ::= SEQUENCE {
    trustpoint              Certificate,                        -- self-signed certificate
    pathLenConstraint   [0] PathLenConstraint   OPTIONAL,
    acceptablePolicySet [1] AcceptablePolicySet OPTIONAL,  -- If not present "any policy"
    nameConstraints     [2] NameConstraints     OPTIONAL,
    policyConstraints   [3] PolicyConstraints   OPTIONAL }

PathLenConstraint    ::=   INTEGER (0..MAX)

AcceptablePolicySet ::= SEQUENCE OF CertPolicyId

CertPolicyId ::= OBJECT IDENTIFIER

NameConstraints ::= SEQUENCE {
           permittedSubtrees       [0]     GeneralSubtrees OPTIONAL,
           excludedSubtrees        [1]     GeneralSubtrees OPTIONAL }

      GeneralSubtrees ::= SEQUENCE SIZE (1..MAX) OF GeneralSubtree

      GeneralSubtree ::= SEQUENCE {
           base                    GeneralName,
           minimum         [0]     BaseDistance DEFAULT 0,
           maximum         [1]     BaseDistance OPTIONAL }

      BaseDistance ::= INTEGER (0..MAX)

PolicyConstraints ::= SEQUENCE {
        requireExplicitPolicy           [0] SkipCerts OPTIONAL,
        inhibitPolicyMapping            [1] SkipCerts OPTIONAL }

SkipCerts ::= INTEGER (0..MAX)

CertRevReq ::= SEQUENCE {
    endCertRevReq   RevReq,
    caCerts         [0] RevReq
      }

RevReq ::= SEQUENCE  {
    enuRevReq  EnuRevReq,
    exRevReq    SignPolExtensions OPTIONAL}

EnuRevReq  ::= ENUMERATED {
    clrCheck    (0), --Checks shall be made against current CRLs
            -- (or authority revocation lists)
    ocspCheck   (1), -- The revocation status shall be checked
            -- using the Online Certificate Status Protocol (RFC 2450)
    bothCheck   (2),    -- Both CRL and OCSP checks shall be carried out
    eitherCheck (3),    -- At least one of CRL or OCSP checks shall be carried out
    noCheck     (4),    -- no check is mandated
    other       (5)     -- Other mechanism as defined by signature poilicy extension
      }

SigningCertTrustCondition ::=   SEQUENCE {
     signerTrustTrees             CertificateTrustTrees,
```

```
     signerRevReq                  CertRevReq
}

TimestampTrustCondition ::= SEQUENCE {
    ttsCertificateTrustTrees    [0]     CertificateTrustTrees    OPTIONAL,
    ttsRevReq                   [1]     CertRevReq              OPTIONAL,
    ttsNameConstraints          [2]     NameConstraints         OPTIONAL,
    cautionPeriod               [3]     DeltaTime               OPTIONAL,
    signatureTimestampDelay     [4]     DeltaTime               OPTIONAL }

DeltaTime ::= SEQUENCE {
    deltaSeconds    INTEGER,
    deltaMinutes    INTEGER,
    deltaHours      INTEGER,
    deltaDays       INTEGER }

AttributeTrustCondition ::= SEQUENCE {
    attributeMandated            BOOLEAN,              -- Attribute shall be present
    howCertAttribute             HowCertAttribute,
    attrCertificateTrustTrees   [0] CertificateTrustTrees   OPTIONAL,
    attrRevReq                  [1] CertRevReq             OPTIONAL,
    attributeConstraints        [2] AttributeConstraints   OPTIONAL }


HowCertAttribute ::= ENUMERATED {
    claimedAttribute    (0),
    certifiedAttribtes  (1),
    either              (2) }

AttributeConstraints ::= SEQUENCE {
    attributeTypeConstarints    [0] AttributeTypeConstraints  OPTIONAL,
    attributeValueConstarints   [1] AttributeValueConstraints OPTIONAL }


AttributeTypeConstraints ::= SEQUENCE OF AttributeType

AttributeValueConstraints ::= SEQUENCE OF AttributeTypeAndValue


AlgorithmConstraintSet ::= SEQUENCE {   -- Algorithm constrains on:
signerAlgorithmConstraints  [0]     AlgorithmConstraints OPTIONAL, -- signer
eeCertAlgorithmConstraints  [1]     AlgorithmConstraints OPTIONAL, -- issuer of end entity certs.
caCertAlgorithmConstraints  [2]     AlgorithmConstraints OPTIONAL, -- issuer of CA certificates
aaCertAlgorithmConstraints  [3]     AlgorithmConstraints OPTIONAL, -- Attribute Authority
tsaCertAlgorithmConstraints [4]     AlgorithmConstraints OPTIONAL  -- TimeStamping Authority
             }

AlgorithmConstraints ::= SEQUENCE OF AlgAndLength

AlgAndLength ::= SEQUENCE {
    algID           OBJECT IDENTIFIER,
    minKeyLength    INTEGER     OPTIONAL, -- Minimum key length in bits
    other       SignPolExtensions OPTIONAL
         }

SignPolExtensions ::= SEQUENCE OF SignPolExtn

SignPolExtn ::= SEQUENCE {
        extnID      OBJECT IDENTIFIER,
    extnValue   OCTET STRING  }

END-- ETS-ElectronicSignature-97Syntax
```

Annex B (informative): Example Structured Contents and MIME

B.1 General Description

The signed content may be structured as using MIME (Multipurpose Internet Mail Extensions - RFC 2045 [27] which obsoletes RFC 1521 [28]). Whilst the MIME structure was initially developed for Internet e-mail, it has a number of features which 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:

- it provides a means of signalling the type of “object" being carried (e.g. text, image, ZIP file, application data);
- it provides a means of associating a file name with an object;
- it can associate several independent “objects" (e.g. a document and image) to form a multi-part object;
- it can handle data encoded in text or binary and, if necessary, re-encode 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.

B.2 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 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 word document is:

```
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64
Content-Description: JCFV201.doc (Microsoft Word Document)
Content-Disposition: filename="JCFV201.doc"
```

B.3 Content Encoding

MIME supports a range of mechanisms for encoding the both text and binary data.

Text data can be carried transparently as lines of text data encoded in 7 or 8 bit ACSII characters. MIME also includes a "quoted-printable" encoding which converts characters other than the basic ASCII into an ACSII sequence.

Binary can either be carried:

- transparently a 8 bit octets; or
- converted to a basic set of characters using a system called Base64.

NOTE: As there are some mail relays which can only handle 7 bit ACSII, Base64 encoding is usually used on the Internet.

B.4 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 indicate 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:

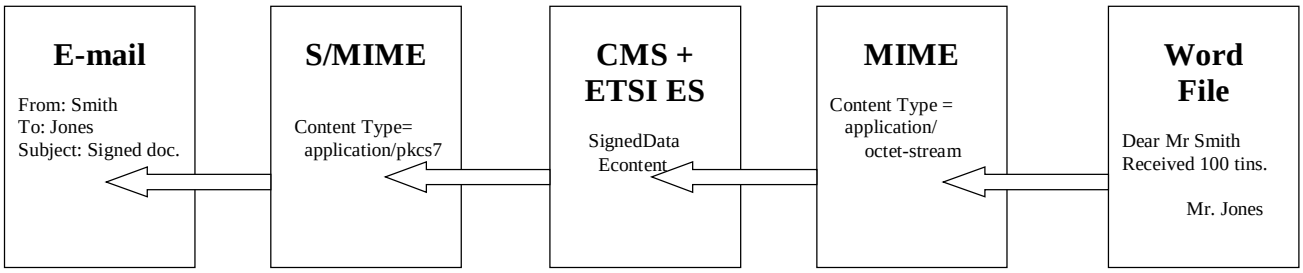
```
Mime-Version: 1.0
Content-Type: multipart/mixed; boundary="----=_NextPart_000_01BC4599.98004A80"
Content-Transfer-Encoding: 7bit

------=_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/octet-stream; name="JCFV201.doc"
Content-Transfer-Encoding: base64
Content-Description: JCFV201.doc (Microsoft Word Document)
Content-Disposition: attachment; filename="JCFV201.doc"

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).

B.5 S/MIME

Previous clauses in this annex have described the use of MIME to encode data. MIME encoded data can be signed (i.e. carried in the eContent of the SignedData structure) thereby signalling the type of information that has been signed.

MIME can also be used to encode the CMS structure containing data after it has been signed so that, for example, this can be carried within an e-mail message. The specific use of MIME to carry CMS (extended as defined in the present document) secured data is called S/MIME. The relationship between the general use of MIME for encoding content, CMS and S/MIME is illustrated in the following diagram:



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)

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
```

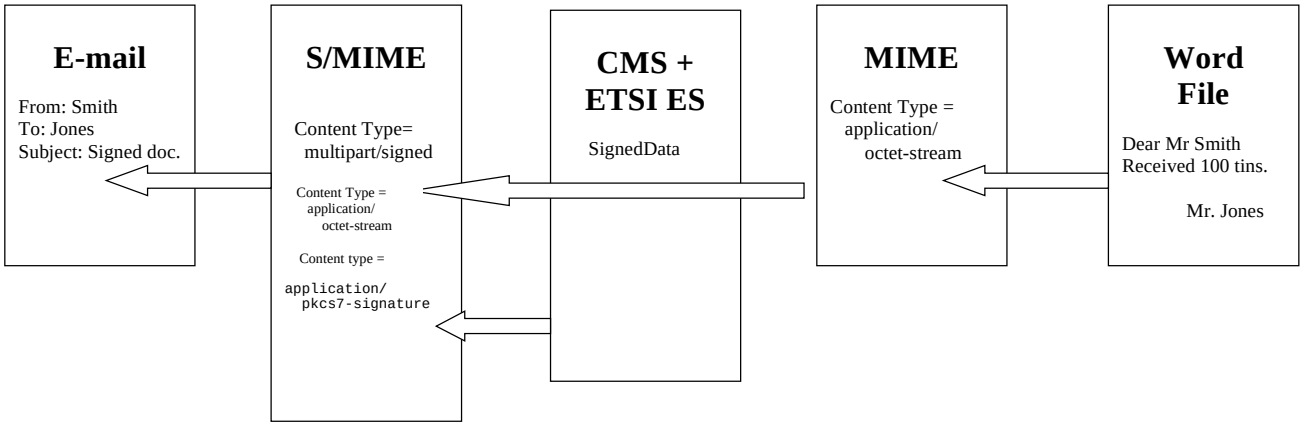
This approach is similar to handling signed data as any other binary file attachment. Thus, this encoding can be used where signed data passes through gateways to other e-mail systems (e.g. those based on ITU-T Recommendation X.400 [14] or proprietary e-mail systems).

A "multipart/signed" object with the signed data and the signature encoded as separate MIME objects.

An example of signed data encoded 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 MIME the signed data passes through the CMS process and is carried as part of the S/MIME structure as illustrated in the following diagram. The CMS structure just holds the electronic signature.



The second approach (multipart/signed) has the advantage that the signed data can be decoded by any MIME compatible e-mail system even if it doesn't recognize CMS encoded electronic signatures. However, this form cannot be used with other e-mail systems.

Annex C (informative): Relationship to the European Directive and EESSI

C.1 Introduction

This annex provides an indication of the relationship between electronic signatures created under the present document and requirements under the European Parliament and Council Directive on a Community framework for electronic signatures.

NOTE: Legal advice should be sought on the specific national legislation regarding use of electronic signatures.

This standard is one of a set of standards being defined under the “European Electronic Signature Standardization Initiative" (EESSI) for electronic signature products and solutions compliant with the European Directive for electronic signatures.

C.2 Electronic Signatures and the Directive

This directive defines electronic signatures as:

"data in electronic form which are attached to or logically associated with other electronic data and which serve as a method of authentication".

The directive states that an electronic signature should not be denied "legal effectiveness and admissibility as evidence in legal proceedings" solely on the grounds that it is in electronic form.

The directive identifies an electronic signature as having equivalence to a hand-written signature if it meets specific criteria:

- it is an "advanced electronic signature" with the following properties:

a) it is uniquely linked to the signatory;

b) it is capable of identifying the signatory;

c) it is created using means that the signatory can maintain under his sole control; and

d) it is linked to the data to which it relates in such a manner that any subsequent change of the data is detectable.

- it is based on a certificate which meets detailed criteria given in annex I to the directive and is issued by a "certification-service-provider" which meets requirements given annex II to the directive. Such a certificate is referred to as a "qualified certificate";
- it is created by a "device" which detailed criteria given in annex III to the directive. Such a device is referred to a "secure-signature-creation device";

This form of electronic signature is referred to as a "qualified electronic signature" in EESSI (see below)..

C.3 ETSI Electronic Signature Formats and the Directive

An electronic signature created in accordance with the present document is:

a) considered to be an "electronic signature" under the terms of the Directive;

b) considered to be an "advanced electronic signature" under the terms of the Directive;

c) considered to be a "Qualified Electronic Signature" provided the additional requirements in annex I, II and III of the Directive are met. The requirements in annex I, II and III of the Directive are outside the scope of the present document, and are subject to further standardization..

C.4 EESSI Standards and Classes of Electronic Signature

C.4.1. Structure of EESSI standardtion

EESSI looks at standards in the following areas:

- use of X.509 public key certificates as qualified certificates;
- security Management and Certificate Policy for CSPs Issuing Qualified Certificates;
- security requirements for trustworthy systems used by CSPs Issuing Qualified Certificates;
- security requirements for signature creation devices;
- signature creation and verification;
- electronic signature syntax and encoding formats;
- technical aspects of signature policies;
- protocol to interoperate with a Time Stamping Authority.

Each of these standards shall address a range of requirements including the requirements of Qualified Electronic Signatures as specified in article 5.1 of the Directive. However, it shall also address general requirements of electronic signatures for business and electronic commerce which all fall into the category of article 5.2 of the Directive. Such variation in the requirements may be identified in the standard either as different levels or different options.

C.4.2 Classes of electronic signatures

Since each standard addresses a range of requirements, it will be necessary to identify a set of standards and the use of each standard, "profiles", to address a specific business need. Such a set of standards and their uses defines a **class of electronic signature**. One of the first classes to be defined is the qualified electronic signature, fulfilling the requirements of 5.1 of the Directive.

A limited number of "classes of electronic signatures" and corresponding profiles should be defined by EESSI, in close co-operation with actors on the market (business, users, suppliers). Need for standards is envisaged, in addition to those for qualified electronic signatures, in areas such as:

- electronic signatures with long term validity;
- electronic signatures for business transactions with limited value.

C.4.3 EESSI Classes and the ETSI Electronic Signature Format

The electronic signature format defined in the present document is applicable to the EESSI area "electronic signature and encoding formats".

An electronic signature produced by a signer (see clause 8 and conformance subclause 14.1) is applicable to the proposed class of electronic signature: "qualified electronic signatures fulfilling article 5.1".

With the addition of validation data by the verifier (see clause 9 and conformance subclause 14.2) this would become applicable to a new class of electronic signature adding a long-term validity attribute to the qualified electronic signature.

Annex D (informative): APIs for the Generation and Verification of Electronic Signatures Tokens

While the present document describes the data format of an electronic signature, the question is whether there exists APIs (Application Programming Interfaces) able to manipulate these structures. At least two such APIs have been defined. One set by the IETF and another set by the OMG (Object Management Group).

D.1 Data Framing

In order to be able to use either of these APIs, it will be necessary to frame the previously defined electronic signature data structures using an mechanism-independent token format. Section 3.1 of RFC 2078 [29] describes that framing incorporating an identifier of the mechanism type to be used and enabling tokens to be interpreted unambiguously.

In order to be processable by these APIs, all electronic signature data formats that are defined in the present document shall be framed following that description.

The encoding format for the token tag is derived from ASN.1 and DER, but its concrete representation is defined directly in terms of octets rather than at the ASN.1 level in order to facilitate interoperable implementation without use of general ASN.1 processing code. The token tag consists of the following elements, in order:

1) 0x60 -- Tag for [APPLICATION 0] SEQUENCE; indicates that constructed form, definite length encoding follows.

2) Token length octets, specifying length of subsequent data (i.e., the summed lengths of elements 3-5 in this list, and of the mechanism-defined token object following the tag). This element comprises a variable number of octets:

   - If the indicated value is less than 128, it shall be represented in a single octet with bit 8 (high order) set to "0" and the remaining bits representing the value.

   - If the indicated value is 128 or more, it shall be represented in two or more octets, with bit 8 of the first octet set to "1" and the remaining bits of the first octet specifying the number of additional octets. The subsequent octets carry the value, 8 bits per octet, most significant digit first. The minimum number of octets shall be used to encode the length (i.e. no octets representing leading zeros shall be included within the length encoding).

3) 0x06 -- Tag for OBJECT IDENTIFIER.

4) Object identifier length -- length (number of octets) of the encoded object identifier contained in element 5, encoded per rules as described in 2a. and 2b. above.

5) object identifier octets -- variable number of octets, encoded per ASN.1 BER rules:

   - The first octet contains the sum of two values: (1) the top-level object identifier component, multiplied by 40 (decimal), and (2) the second-level object identifier component. This special case is the only point within an object identifier encoding where a single octet represents contents of more than one component.

   - Subsequent octets, if required, encode successively-lower components in the represented object identifier. A component's encoding may span multiple octets, encoding 7 bits per octet (most significant bits first) and with bit 8 set to "1" on all but the final octet in the component's encoding. The minimum number of octets shall be used to encode each component (i.e. no octets representing leading zeros shall be included within a component's encoding).

NOTE: In many implementations, elements 3 to 5 may be stored and referenced as a contiguous string constant.

The token tag is immediately followed by a mechanism-defined token object. Note that no independent size specifier intervenes following the object identifier value to indicate the size of the mechanism- defined token object.

Tokens conforming to the present document shall have the following OID in order to be processable by IDUP-APIs:

```
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) }
```

D.2 IDUP-GSS-APIs defined by the IETF

The IETF CAT WG has produced in December 1998 an RFC (RFC 2479 [30]) under the name of IDUP-GSS-API (Independent Data Unit Protection) able to handle the electronic signature data format defined in the present document.

The IDUP-GSS-API includes support for non-repudiation services. It supports evidence generation, where "evidence" is information that either by itself, or when used in conjunction with other information, is used to establish proof about an event or action, as well a evidence verification.

IDUP supports various types of evidences. All the types defined in IDUP are supported in the present document through the commitment type parameter.

The section 2.3.3. of IDUP describes the specific calls needed to handle evidences ("EV" calls). The "EV" group of calls provides a simple, high-level interface to underlying IDUP mechanisms when application developers need to deal only with evidences but not with encryption or integrity services.

All generations and verification are performed according to the content of a **NR policy** that is referenced in the context.

Get_token_details is used to return to an application the attributes that correspond to a given input token. Since IDUP-GSS- API tokens are meant to be opaque to the calling application, this function allows the application to determine information about the token without having to violate the opaqueness intention of IDUP. Of primary importance is the mechanism type, which the application can then use as input to the **IDUP_Establish_Env**() call in order to establish the correct environment in which to have the token processed.

Generate_token generates a non-repudiation token using the current environment.

Verify_evidence verifies the evidence token using the current environment. This operation returns a major_status code which can be used to determine whether the evidence contained in a token is complete (i.e., can be successfully verified (perhaps years) later). If a token's evidence is not complete, the token can be passed to another API: form_complete_pidu to complete it. This happens when a status "conditionally valid" is returned. That status corresponds to the status "validation incomplete" of the present document.

Form_complete_PIDU is used primarily when the evidence token itself does not contain all the data required for its verification and it is anticipated that some of the data not stored in the token may become unavailable during the interval between generation of the evidence token and verification unless it is stored in the token. The Form_Complete_PIDU operation gathers the missing information and includes it in the token so that verification can be guaranteed to be possible at any future time.

D.3 CORBA Security interfaces defined by the OMG

Non-repudiation interfaces have been defined in "CORBA Security", a document produced by the OMG (Object Management Group). These interfaces are described in IDL (Interface Definition Language) and are optional.

The handling of "tokens" supporting non-repudiation is done through the following interfaces:

- **set_NR_features** specifies the features to apply to future evidence generation and verification operations.
- **get_NR_features** returns the features which will be applied to future evidence generation and verification operations.
- **generate_token** generates a Non-repudiation token using the current Non-repudiation features.
- **verify_evidence** verifies the evidence token using the current Non-repudiation features.

- **get_tokens-details** returns information about an input Non-repudiation token. The information returned depends upon the type of token.

- **form_complete_evidence** is used when the evidence token itself does not contain all the data required for its verification, and it is anticipated that some of the data not stored in the token may become unavailable during the interval between generation of the evidence token and verification unless it is stored in the token. The form_complete_evidence operation gathers the missing information and includes it in the token so that verification can be guaranteed to be possible at any future time.

NOTE: The similarity between the two sets of APIs is noticeable.

Annex E (informative): Cryptographic Algorithms

E.1 Digest Algorithms

Section 12.1 of RFC 2630 [9] states that SHA-1and MD5 following that shall be supported for use with CMS.

E.1.1 SHA-1

The SHA-1 digest algorithm is defined in FIPS Pub 180-1. The algorithm identifier for SHA-1 is:

```
sha-1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 26 }
```

The AlgorithmIdentifier parameters field is optional. If present, the parameters field shall contain an ASN.1 NULL. Implementations should accept SHA-1 AlgorithmIdentifiers with absent parameters as well as NULL parameters. Implementations should generate SHA-1 AlgorithmIdentifiers with NULL parameters.

E.1.2 MD5

The MD5 digest algorithm is defined in RFC 1321 [23]. The algorithm identifier for MD5 is:

```
md5 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) digestAlgorithm(2) 5 }
```

The AlgorithmIdentifier parameters field shall be present, and the parameters field shall contain NULL. Implementations may accept the MD5 AlgorithmIdentifiers with absent parameters as well as NULL parameters.

E.1.3 General

The following is a selection of work that has been done in the area of digest algorithms or, as they are often called, hash functions:

- ISO/IEC 10118-1 [33] (1994): "Information technology - Security techniques - Hash-functions - Part 1: General". ISO/IEC 10118-1 [33] contains definitions and describes basic concepts.
- ISO/IEC 10118-2 [34] (1994): "Information technology - Security techniques - Hash-functions - Part 2: Hash-functions using an n-bit block cipher algorithm". ISO/IEC 10118-2 [34] specifies two ways to construct a hash-function from a block cipher.
- ISO/IEC 10118-3 [35] (1997): "Information technology - Security techniques - Hash-functions - Part 3: Dedicated hash-functions". ISO/IEC 10118-3 [35] specifies the following dedicated hash-functions:
  - SHA-1 (FIPS 180-1)
  - RIPEMD-128
  - RIPEMD-160.
- ISO/IEC FCD 10118-4 [36]: "Information technology - Security techniques - Hash-functions - Part 4: Hash-functions using modular arithmetic". Status: Final Committee Draft; Expected publication date: 1998 ISO/IEC 10118-4 specifies ways to construct a hash-function from a modular multiplication.
- Internet RFC 1320 [38] (PS 1992): "The MD4 Message-Digest Algorithm". RFC 1320 [38] specifies the hash-function MD4. Today, MD4 is considered out-dated.
- Internet RFC 1321 [23] (I 1992): "The MD5 Message-Digest Algorithm". RFC 1321 [23] (informational) specifies the hash-function MD5.

- FIPS Publication 180-1 (1995): "Secure Hash Standard". FIPS 180-1 [39] specifies the Secure Hash Algorithm (SHA), dedicated hash-function developed for use with the DSA. The original SHA published in 1993 was slightly revised in 1995 and renamed SHA-1.
- ANS X9.30-2 [31] (1997): "Public Key Cryptography for the Financial Services Industry - Part 2: The Secure Hash Algorithm (SHA-1)". X9.30-2 specifies the ANSI-Version of SHA-1.
- ANS X9.31-2 [40] (draft): "Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry - Part 2: Hash Algorithms". X9.31-2 specifies hash algorithms.

E.2 Digital Signature Algorithms

Section 12.2 of RFC 2630 [9] states that CMS implementations shall include DSA and may include RSA.

E.2.1 DSA

The DSA signature algorithm is defined in FIPS Pub 186. DSA is always used with the SHA-1 message digest algorithm. The algorithm identifier for DSA is:

```
id-dsa-with-sha1 OBJECT IDENTIFIER ::=  { iso(1) member-body(2) us(840) x9-57 (10040) x9cm(4) 3 }
```

The AlgorithmIdentifier parameters field shall not be present.

E.2.2 RSA

The RSA signature algorithm is defined in RFC 2437 [32]. RFC 2437 [32] specifies the use of the RSA signature algorithm with the SHA-1 and MD5 message digest algorithms. The algorithm identifier for RSA is:

```
rsaEncryption OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549)
   pkcs(1) pkcs-1(1) 1 }
```

E.2.3 General

The following is a selection of work that has been done in the area of digital signature mechanisms:

- FIPS Publication 186 (1994): "Digital Signature Standard". NIST's *Digital Signature Algorithm* (DSA) is a variant of ElGamal's Discrete Logarithm based digital signature mechanism. The DSA requires a 160-bit hash-function and mandates SHA-1.
- IEEE P1363: "Standard Specifications for Public-Key Cryptography". Status: Draft, Expected publication date: 1999. The current draft contains mechanisms for digital signatures, key establishment, and encipherment based on three families of public-key schemes:
  - "Conventional" Discrete Logarithm (DL) based techniques, i.e., Diffie-Hellman (DH) key agreement, Menezes-Qu-Vanstone (MQV) key agreement, the *Digital Signature Algorithm* (DSA), and Nyberg-Rueppel (NR) digital signatures.
  - Elliptic Curve (EC) based variants of the DL-mechanisms specified above, i.e., EC-DH, EC-MQV, EC-DSA, and EC-NR. For elliptic curves, implementation options include mod p and characteristic 2 with polynomial or normal basis representation.
  - Integer Factoring (IF) based techniques including RSA encryption, RSA digital signatures, and RSA-based key transport.
- ISO/IEC 9796 [43] (1991): "Information technology - Security techniques - Digital signature scheme giving message recovery". ISO/IEC 9796 [43] specifies a digital signature mechanism based on the RSA public-key technique and a specifically designed redundancy function.

- ISO/IEC 9796-2 [44] (1997): "Information technology - Security techniques - Digital signature schemes giving message recovery - Part 2: Mechanisms using a hash-function". ISO/IEC 9796-2 [44] specifies digital signature mechanisms with partial message recovery that are also based on the RSA technique but make use of a hash-function.

- ISO/IEC CD 9796-4 [45]: "Digital signature schemes giving message recovery - Part 4: Discrete logarithm based mechanisms". Status: Committee Draft; Expected publication date: 2000. ISO/IEC 9796-4 [45] specifies digital signature mechanisms with partial message recovery that are based on Discrete Logarithm techniques. The current draft includes the Nyberg-Rueppel scheme.

- ISO/IEC FCD 14888-1 [46]: "Digital signatures with appendix - Part 1: General". Status: Final Committee Draft; Expected publication date: 1999. ISO/IEC 14888-1 [46] contains definitions and describes the basic concepts of digital signatures with appendix.

- ISO/IEC FCD 14888-2 [47]: "Digital signatures with appendix - Part 2: Identity-based mechanisms". Status: Final Committee Draft; Expected publication date: 1999. ISO/IEC 14888-2 [47] specifies digital signature schemes with appendix that make use of identity-based keying material. The current draft includes the zero-knowledge techniques of Fiat-Shamir and Guillou-Quisquater.

- ISO/IEC FCD 14888-3 [48]: "Digital signatures with appendix - Part 3: Certificate-based mechanisms". Status: Final Committee Draft; Expected publication date: 1999. ISO/IEC 14888-3 [48] specifies digital signature schemes with appendix that make use of certificate-based keying material. The current draft includes five schemes:

  - DSA;
  - EC-DSA, an elliptic curve based analog of NIST's Digital Signature Algorithm;
  - Pointcheval-Vaudeney signatures;
  - RSA signatures;
  - ESIGN.

- ISO/IEC WD 15946-2 [49]: "Cryptographic techniques based on elliptic curves - Part 2: Digital signatures". Status: Working Draft; Expected publication date: 2000. ISO/IEC 15946-3 specifies digital signature schemes with appendix using elliptic curves. The current draft includes two schemes:

  - EC-DSA, an elliptic curve based analog of NIST's Digital Signature Algorithm;
  - EC-AMV. an elliptic curve based analog of the Agnew-Muller-Vanstone signature algorithm.

- ANS X9.31-1 [50] (draft): "Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry - Part 1: The RSA Signature Algorithm". ANSI X9.31-1 specifies a digital signature mechanism with appendix using the RSA public-key technique.

- ANS X9.30-1 [51] (1997): "Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry - Part 1: The Digital Signature Algorithm (DSA)". ANSI X9.30-1 specifies the DSA, NIST's *Digital Signature Algorithm*.

- ANS X9.62 [52] (draft): "Public Key Cryptography for the Financial Services Industry - The Elliptic Curve Digital Signature Algorithm (ECDSA)". The ANSI X9.62 draft standard specifies the *Elliptic Curve Digital Signature Algorithm*, an analog of NIST's *Digital Signature Algorithm* (DSA) using elliptic curves. The appendices provide tutorial information on the underlying mathematics for elliptic curve cryptography and many examples.

Annex F (informative): Guidance on Naming

F.1 Allocation of Names

It is necessary to unambiguously identify the subject of a certificate. This requires the applicant (subject applying for a certificate) to be given a name which uniquely identifies him/her, before issuing the certificate. Thus, the subject name shall be allocated through a registration scheme administered through a Registration Authority (RA) to ensure uniqueness. This RA may be an independent body or a function carried out by the Certification Authority.

In addition to ensuring uniqueness, the RA shall verify that the name allocated properly identifies the applicant and that authentication checks are carried out to protect against masquerade.

The name allocated by an RA is based on registration information provided by, or relating to, the applicant (e.g. his personal name, date of birth, residence address) and information allocated by the RA. Three variations commonly exist:

- the name is based entirely on registration information which uniquely identifies the applicant (e.g. "Pierre Durand (born on) July 6, 1956");
- the name is based on registration information with the addition of qualifiers added by the registration authority to ensure uniqueness (e.g. "Pierre Durand 12");
- the registration information is kept private by the registration authority and the registration authority allocates a "pseudonym".

F.2 Providing Access to Registration Information

Under certain circumstances it may be necessary for information used during registration, but not published in the certificate, to be made available to third parties (e.g. to an arbitrator to resolve a dispute or for law enforcement). This registration information is likely to include personal and sensitive information.

Thus the RA needs to establish a policy for:

- whether the registration information should be disclosed;
- to whom such information should be disclosed;
- under what circumstances such information should be disclosed.

This policy may be different whether the RA is being used only within a company or for public use. The policy will have to take into account national legislation and in particular any data protection and privacy legislation.

Currently, the provision of access to registration is a local matter for the RA. However, if open access is required, standard protocols such as HTTP - RFC 2068 [25] (Internet Web Access Protocol) may be employed with the addition of security mechanisms necessary to meet the data protection requirements (e.g. Transport Layer Security - RFC 2246 [24] with client authentication).

F.3 Naming Schemes

F.3.1 Naming Schemes for Individual Citizens

In some cases the subject name that is contained in a public key certificate may not be meaningful enough. This may happen because of the existence of homonyms or because of the use of pseudonyms. A distinction could be made if more attributes were present. However, adding more attributes to a public key certificate placed in a public repository would be going against the privacy protection requirements. In any case the Registration Authority will get information at the time of registration but not all that information will be placed in the certificate. In order to achieve a balance between these two opposite requirements the hash values of some additional attributes can be placed in a public key certificate. When the certificate owner provides these additional attributes, then they can be verified. Using biometrics attributes may unambiguously identify a person. Example of biometrics attributes that can be used include: a picture or a manual signature from the certificate owner.

NOTE: Using hash values protects privacy only if the possible inputs are large enough. For example, using the hash of a person's social security number is generally not sufficient since it can easily be reversed.

A picture can be used if the verifier once met the person and later on wants to verify that the certificate that he or she got relates to the person whom was met. In such a case, at the first exchange the picture is sent and the hash contained in the certificate may be used by the verifier to verify that it is the right person. At the next exchange the picture does not need to be sent again. A manual signature may be used if a signed document has been received beforehand. In such a case, at the first exchange the drawing of the manual signature is sent and the hash contained in the certificate may be used by the verifier to verify that it is the right manual signature. At the next exchange the manual signature does not need to be sent again.

F.3.2 Naming Schemes for Employees of an Organization

The name of an employee within an organization is likely to be some combination of the name of the organization and the identifier of the employee within that organization.

An organization name is usually a *registered* name, i.e. business or trading name used in day to day business. This name is registered by a Naming Authority, which guarantees that the organization's registered name is unambiguous and cannot be confused with another organization. In order to get more information about a given *registered organization name*, it is necessary to go back to a publicly available directory maintained by the Naming Authority.

The identifier may be a name or a pseudonym (e.g. a nickname or a employee number). When it is a name, it is supposed to be descriptive enough to unambiguously identify the person. When it is a pseudonym, the certificate does not disclose the identity of the person. However it ensures that the person has been correctly authenticated at the time of registration and therefore may be eligible to some advantages implicitly or explicitly obtained through the possession of the certificate. In either case, however, this can be insufficient because of the existence of homonyms.

Placing more attributes in the certificate may be one solution, for example by giving the organization unit of the person or the name of a city where the office is located. However the more information is placed in the certificate the more problems arise if there is a change in the organization structure or the place of work. So this may not be the best solution. An alternative is to provide more attributes (like the organization unit and the place of work) through access to a directory maintained by the company. It is likely that at the time of registration the Registration Authority got more information than what was placed in the certificate, if such additional information is placed in a repository accessible only to the organization.

Annex G (informative): Attribute Certificate Format

G.1 Attribute certificate structure

An attribute certificate is a separate structure from a subject's public key certificate. A subject may have multiple attribute certificates associated with each of its public key certificates. There is no requirement that the same authority create both the public key certificate and attribute certificate(s) for a user; in fact separation of duties will frequently dictate otherwise. In environments where different authorities have responsibility for issuing public key and attribute certificates, the public key certificate(s) issued by a Certification Authority (CA) and the attribute certificate(s) issued by an Attribute Authority (AA) would be signed using different private signing keys. In environments where a single entity is both the CA, issuing public key certificates, and the AA, issuing attribute certificates, it is strongly recommended that a different key be used to sign attribute certificates than the key used to sign public-key certificates. Exchanges between the issuing authority and the entity receiving a certificate are outside the scope of the present document.

The attribute certificate is defined in ITU-T Recommendation X.509: "Draft Amendment on Certificate Extensions", October 1999 as follows.

```
AttributeCertificate ::= SIGNED {AttributeCertificateInfo}
AttributeCertificateInfo ::= SEQUENCE
    {
    version             AttCertVersion DEFAULT v1,
    holder              Holder,
    issuer              AttCertIssuer,
    signature           AlgorithmIdentifier,
    serialNumber        CertificateSerialNumber,
    attrCertValidityPeriod  AttCertValidityPeriod
    attributes          SEQUENCE OF Attribute,
    issuerUniqueID      UniqueIdentifier OPTIONAL,
    extensions          Extensions  OPTIONAL
    }

AttCertVersion  ::= INTEGER {v1(0), v2(1) }

Holder  ::= SEQUENCE
        {
        baseCertificateID   [0] IssuerSerial    OPTIONAL,
        -- the issuer and serial number of the holder's Public Key Certificate
        entityName      [1] GeneralNames    OPTIONAL,
        -- the name of the entity or role
        objectDigestInfo    [2] ObjectDigestInfo    OPTIONAL
        -- if present, version must be v2
--at least one of baseCertificateID, entityName or objectDigestInfo must be present--}
ObjectDigestInfo    ::= SEQUENCE {
    digestedObjectType  ENUMERATED {
        publicKey           (0),
        publicKeyCert           (1),
        otherObjectTypes     (2) },
    otherObjectTypeID       OBJECT IDENTIFIER  OPTIONAL,
    digestAlgorithm             AlgorithmIdentifier,
    objectDigest                BIT STRING }

AttCertIssuer       ::= SEQUENCE {
        issuerName          GeneralNames    OPTIONAL,
        baseCertificateId   [0] IssuerSerial    OPTIONAL,
        objectDigestInfo    [1] ObjectDigestInfo    OPTIONAL
--at least one of issuerName, baseCertificateId or objectDigestInfo must be present–
-- if baseCertificateId or objectDigestInfo are present, version must be v2-- }
IssuerSerial  ::=  SEQUENCE {
    issuer      GeneralNames,
    serial          CertificateSerialNumber,
    issuerUID   UniqueIdentifier OPTIONAL }
AttCertValidityPeriod  ::= SEQUENCE {
    notBeforeTime   GeneralizedTime,
    notAfterTime    GeneralizedTime }
```

The **version** number differentiates between different versions of the attribute certificate. If holder includes **objectDigestInfo** or if **issuer** includes **baseCertificateID** or **objectDigestInfo**, **version** shall be **v2**.

The **holder** field conveys the identity of the attribute certificate's holder.

The **baseCertificateId** component, if present, identifies a particular public-key certificate that is to be used to authenticate the identity of this holder when asserting privileges with this attribute certificate.

The **entityName** component, if present, identifies one or more names for the holder. If **entityName** is the only component present in **holder**, any public-key certificate that has one of these names as its subject can be used to authenticate the identity of this holder when asserting privileges with this attribute certificate. If **baseCertificateID** and **entityName** are both present, only the certificate specified by **baseCertificateID** may be used. In this case **entityName** is included only as a tool to help the privilege verifier locate the identified public-key certificate.

NOTE 1: There is a risk with the sole use of **GeneralNames** to identify the holder, in that this points only to a name for the holder. This is generally insufficient to enable the authentication of a holder's identity for purposes of issuing privileges to that holder. Use of the issuer name and serial number of a specific public-key certificate, however, enables the issuer of attribute certificates to rely on the authentication process performed by the CA when issuing that particular public-key certificate. Also, some of the options in **GeneralNames** (e.g. **IPAddress**) are inappropriate for use in naming an attribute certificate holder, especially when the holder is a role and not an individual entity. Another problem with GeneralNames alone as an identifier for a holder is that many name forms within that construct do not have strict registration authorities or processes for the assignment of names.

The **objectDigestInfo** component, if present, is used directly to authenticate the identity of a holder, including an executable holder (e.g. an applet). The holder is authenticated by comparing a digest of the corresponding information, created by the privilege verifier with the same algorithm identified in **objectDigestInfo** with the content of **objectDigest**. If the two are identical, the holder is authenticated for purposes of asserting privileges with this attribute certificate.

publicKey shall be indicated when a hash of an entity's public-key is included. Hashing a public-key may not uniquely identify one certificate (i.e. the identical key value may appear in multiple certificates). In order to link an attribute certificate to a public-key the hash shall be calculated over the representation of that public-key which would be present in a public-key certificate. Specifically, the input for the hash algorithm shall be the DER encoding of a **SubjectPublicKeyInfo** representation of the key. Note that this includes the **AlgorithmIdentifier** as well as the **BIT STRING**. Note that if the public-key value used as input to the hash function has been extracted from a public-key certificate, then it is possible (e.g. if parameters for the Digital Signature Algorithm were inherited) that this may not be sufficient input for the HASH. The correct input for hashing in this context will include the value of the inherited parameters and thus may differ from the **SubjectPublicKeyInfo** present in the public-key certificate.

publicKeyCert shall be indicated when a public-key certificate is hashed, the hash is over the entire DER encoding of the public-key certificate, including the signature bits.

otherObjectTypes shall be indicated when objects other than public-keys or public-key certificates are hashed (e.g. software objects). The identity of the type of object may optionally be supplied. The portion of the object to be hashed can be determined either by the explicitly stated identifier of the type or, if the identifier is not supplied, by the context in which the object is used.

The **issuer** field conveys the identity of the AA that issued the certificate.

The **issuerName** component, if present, identifies one or more names for the issuer.

The **baseCertificateId** component, if present, identifies the issuer by reference to a specific public-key certificate for which this issuer is the subject.

The **objectDigestInfo** component, if present, identifies the issuer by providing a hash of identifying information for the issuer.

The **signature** identifies the cryptographic algorithm used to digitally sign the attribute certificate.

The **serialNumber** is the serial number that uniquely identifies the attribute certificate within the scope of its issuer.

The **attrCertValidityPeriod** field conveys the time period during which the attribute certificate is considered valid, expressed in **GeneralizedTime** format.

The **attributes** field contains the attributes associated with the holder that are being certified (e.g. the privileges).

NOTE 2: In the case of attribute descriptor attribute certificates, this sequence of attributes can be empty.

The **issuerUniqueID** may be used to identify the issuer of the attribute certificate in instances where the issuer component is not sufficient.

The **extensions** field allows addition of new fields to the attribute certificate.

The framework for attribute certificates described in this section is primarily focused on the model in which privilege is placed within attribute certificates. However, as mentioned earlier, the certificate extensions defined in this section can also be placed in a public-key certificate using the **subjectDirectoryAttributes** extension.

Bibliography

The following material, though not specifically referenced in the body of the present document (or not publicly available), gives supporting information.

- CCITT Recommendation X.209 (1988): "Specification of basic encoding rules for Abstract Syntax Notation One (ASN.1)".
- ITU-T Recommendation X.681 (1997) | ISO/IEC 8824-2: "Information technology - Abstract Syntax Notation One (ASN.1): Information object specification".
- ITU-T Recommendation X.509: "Draft Amendment on Certificate Extensions".

NOTE: Only used regarding syntax of Attribute Certificate as defined in annex G.

- "European Parliament and Council Directive on a Community framework for electronic signatures".
- PKCS #1 V2.0 (1998): "RSA Cryptography Standard", RSA Laboratories.
- IETF RFC 2527 (1999): "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework".
- IETF RFC 2528 (1999): "Internet X.509 Public Key Infrastructure; Representation of Key Exchange Algorithm (KEA) Keys in Internet X.509 Public Key Infrastructure Certificates".
- IETF RFC 2585 (1999): "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP".
- IETF RFC 2313 (1998): "PKCS 1: RSA Encryption Version, Version 1.5".

See also annex E for reference documents relating to cryptographic algorithms.

The following documents are IETF Internet-Drafts and working documents of the IETF. For up to date information reference should be to the latest versions. Also later versions may of the documents may make the referenced documents below obsolete:

- Certificate Management Messages over CMS.
- Internet X.509 Public Key Infrastructure Representation of Elliptic Curve Digital Signature Algorithm (ECDSA) Keys and Signatures in Internet X.509 Public Key Infrastructure Certificates.
- Certificate Management Messages over CMS.
- Internet X.509 Public Key Infrastructure Time Stamp Protocols.
- Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols.
- Internet X.509 Public Key Infrastructure PKIX Roadmap.
- Internet X.509 Public Key Infrastructure Qualified Certificates.
- Diffie-Hellman Proof-of-Possession Algorithms.
- An Internet AttributeCertificate Profile for Authorization.
- Basic Event Representation Token v1.
- Internet X.509 Public Key Infrastructure Extending trust in non-repudiation tokens in time.
- Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv3.
- Simple Certificate Validation Protocol (SCVP).
- Using HTTP as a Transport Protocol for CMP.
- Using TCP as a Transport Protocol for CMP.

- Limited AttributeCertificate Acquisition Protocol.
- OCSP Extensions.
- Certificate and CRL Profile.
- A String Representation of General Name.
- XML-Signature Requirements.
- XML-Signature Core Syntax.

History

Document history			
V1.1.1	January 2000	Membership Approval Procedure	MV 200012: 2000-01-25 to 2000-03-24