Draft ETSI EN 319 102 V0.2.1 (2013-11)

Electronic Signatures and Infrastructures (ESI); Procedures for Signature Creation and Validation

STABLE DRAFT FOR PUBLIC REVIEW UNTIL 15 JANUARY 2014

Download the template for comments:

http://docbox.etsi.org/ESI/Open/Latest_Drafts/Template-for-comments.doc

Send comments to E-SIGNATURES_COMMENTS@LIST.ETSI.ORG

Reference
DEN/ESI-0019102 (EN)

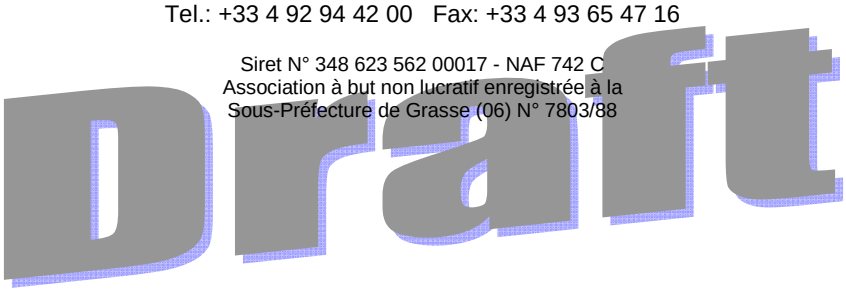
Keywords
electronic signature, security, trust services

ETSI

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

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

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



Important notice

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

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 ETSI 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://ipr.etsi.org).

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

Introduction

As a response to the adoption of Directive 1999/93/EC [] on a Community framework for electronic signatures in 1999, and in order to facilitate the use and the interoperability of eSignature based solution, the European Electronic Signature Standardization Initiative (EESSI) was set up to coordinate the European standardization organisations CEN and ETSI in developing a number of standards for eSignature products.

The European Directive on a community framework for Electronic Signatures [i.15] 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".

To ensure trust in the electronic signature, several aspects must be considered. The different players and the environment of the signature creation and validation have to follow rules to allow them to be trusted. The present document concentrates on policy and security requirements that must be considered when creating and validating signature in a trustworthy manner.

Within the Standardisation Mandate 460[i.14] , issued by the Commission to CEN, CENELEC and ETSI for updating the existing eSignature standardisation deliverables, CEN and ETSI have set up the eSignature Coordination Group in order to coordinate the activities achieved for Mandate 460. The goal of the mandate is to “create the conditions for and achieve the interoperability of eSignature at intra-community level, by defining and providing a rationalised European eSignature standardisation framework” [i.10]. The following web site was set up in the framework in Mandate 460: http://www.e-signatures-standards.eu/.

1 Scope

The present document specifies procedures for

- Creating (Advanced) electronic signatures in a technology-agnostic way. It introduces general principles, objects and functions relevant when creating signatures based on signature creation constraints and defines general forms of advanced electronic signatures that allow verifiability over long periods. It is based on the use of public key cryptography to produce such signatures, which are supported by public key certificates. Such signature creation constraints may be specified as part of a formal signature policy.
- Establishing whether an (Advanced) electronic signature is technically valid based on the considerations specified in the present document and the validation constraints are applied to the verification procedures. These constraints may be specified as part of a formal signature policy.

Clause 4 covers signature creation and

- provides a model of the Signature Creation Environment and a functional model of Signature Creation Applications;
- specifies overall requirements that apply across all of the functions identified in the functional model;

It discusses components of signature creation applications that are intended to deliver to the user or to some other application process in a form specified by the user, an Advanced, or where applicable a Qualified, Electronic Signature associated with a Signer's Document as a Signed Data Object.

Clause 1 introduces the lifecycle of an electronic signature and different forms of advanced electronic signatures that correspond to certain stages of this lifecycle. This includes procedures for upgrading electronic signatures, the process by which certain material (time-stamps, validation data and even archival-related material) is incorporated to the electronic signatures for making them more resilient to change or for enlarging their longevity.

This document is intended to be independent of particular technologies that might be employed in products. The following aspects are considered to be out of scope:

- Generation and distribution of Signature Creation Data (keys etc.), and the selection and use of cryptographic algorithms;
- Format, syntax or encoding of data objects involved, specifically format or encoding for documents to be signed or signatures created;
- The legal interpretation of any form of signature (e.g. the implications of countersignatures, of multiple signatures and of signatures covering complex information structures containing other signatures), specifically whether a signature is accepted by the relying party and if it bears legal validity.

Clause 5 contains an algorithm to validate electronic signatures, with special consideration on signature validation of "old" electronic signatures, where certificates may have expired or been revoked or even the usage period of algorithms have been exceeded. It does so by capitalizing on security measures that have been applied by e.g. the signer or previous verifiers and ensures that such signatures still can be validated. It is agnostic to the type of security measures; while it is primarily aiming at Advanced Electronic Signatures, which provide such features intrinsically, but it also allows for variations, like classical archiving services, where the security measures may also be non-cryptographic.

The way the algorithm is presented aims at clarity and understandability. It is not assumed, nor recommended, that the algorithm will be implemented as described. Efficiency and other implementational aspects were not considered. A conformant implementation will provide the same results, however, as the algorithm here would. An efficient implementation will need to reorder steps in algorithms, use caching of results wherever possible and do things in parallel, if possible.

Signature validation is driven by a signature validation policy. The algorithm presented supports such policies, consisting of a set of policy rules. To avoid confusing terms, the term constraint is used for a single policy rule that influences decisions made by the algorithm. It is assumed that the validator, represented by the driving application, provides such constraints in possibly different forms:

- as a formal signature policy, as specified in [i.3], providing a set of constraints

- as a set of configuration parameters
- by the way the algorithm has been implemented

. Such a formal signature policy may be used exclusively or may be combined with other constraints.

NOTE 1: Factors outside the scope of the present document, such as delays in reporting revocations or unintended data errors in a document, may impact on the signature and so may need to be taken into account in considering the technical validity of a signature in case of dispute.

NOTE 2: The present document makes use of certain verbal forms (e.g. may, shall, shall not and should) as key words to signify requirements, conforming to ETSI Drafting Rules, clause 14a [i.8].

2 References

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

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

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

2.1 Normative references

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

[1] ETSI EN 319 132: “Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES)”

[2] ETSI EN 319 122: “Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES)”

[3] ETSI TS 102 231: "Electronic Signatures and Infrastructures (ESI); Provision of harmonized Trust-service status Information".

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

[5] ETSI TS 101 862: "Qualified certificate Profile".

[6] ISO/IEC 9594-8:2008: "Information technology -- Open Systems Interconnection -- The Directory: Public-key and attribute certificate frameworks".

[7] ETSI TS 101 456: "Electronic Signatures and Infrastructures (ESI); Policy requirements for certification authorities issuing qualified certificates".

[8] ETSI TS 102 042: "Electronic Signatures and Infrastructures (ESI); Policy requirements for certification authorities issuing public key certificates".

[9] Not used.

[10] W3C Recommendation (2008): "XML Signature Syntax and Processing".

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

[12] ETSI EN 319 142-1: “Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signatures (PAdES)”

[13] ETSI TS EN 319 142-3: "Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 3: PAdES Enhanced - PAdES-BES and PAdES-EPES Profiles".

[14] ETSI EN 319 142--4: "Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 4: PAdES Long Term - PAdES LTV Profile".

[15] ETSI EN 319 142--5: "Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 5: PAdES for XML Content - Profiles for XAdES signatures".

[16] IETF RFC 3852: "Cryptographic Message Syntax (CMS)".

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

[18] ETSI TS 103 171: "Electronic Signatures and Infrastructures (ESI); XAdES Baseline Profile".

[19] ETSI TS 103 172: "Electronic Signatures and Infrastructures (ESI); PAdES Baseline Profile".

[20] ETSI TS 103 173: "Electronic Signatures and Infrastructures (ESI); CAdES Baseline Profile".

[21] IETF RFC 5035: “Enhanced Security Services (ESS) Update”

[22] ISO/IEC 7816-15 – ‘Cryptographic Token Information for IC Cards’

[23] PKCS #15: Cryptographic Token Information Format Standard, RSA Laboratories.

NOTE: The documents [1], [2], [12], [13], [14], [15] are published in the context of the work in Mandate M460. They might not yet be published.

2.2 Informative references

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

[i.1] IETF RFC 4158: "Internet X.509 Public Key Infrastructure: Certification Path Building".

[i.2] ETSI TR 102 272: "Electronic Signatures and Infrastructures (ESI); ASN.1 format for signature policies".

[i.3] ETSI TR 102 038: "TC Security - Electronic Signatures and Infrastructures (ESI); XML format for signature policies".

[i.4] "Certificate Validation: back to the past", Moez Ben MBarka and Julien Stern, EuroPKI 2011, 15-16 September 2011, Leuven - Belgium.

[i.5] ECRYPT II Yearly Report on Algorithms and Keysizes (2010-2011), Revision 1.0, 30. June 2011.

[i.6] Commission Decision 2009/767/EC amended by Commission Decision 2010/425/EU.

[i.7] Directive 2006/123/EC of the European Parliament and of the Council of 12 December 2006 on services in the internal market.

[i.8] ETSI Drafting Rules (EDRs).

NOTE: Contained in the ETSI Directives: http://portal.etsi.org/Directives/home.asp.

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

[i.10] ETSI SR 001 604: "Rationalised Framework for Electronic Signature Standardisation".

[i.11] CWA 14170, CEN Workshop Agreement: Security Requirements for Signature creation Applications, May 2004.

[i.12] ETSI TS 119 312 Cryptographic Suites for Secure Electronic Signatures;

[i.13] IETF RFC 6277: " Online Certificate Status Protocol Algorithm Agility".

[i.14] Mandate M460: "Standardisation Mandate to the European Standardisation Organisations CEN, CENELEC and ETSI in the Field of Information and Communication Technologies Applied to Electronic Signatures".

[i.15] Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures.

[i.16] ETSI TS 119 612 Trusted Lists;

3 Definitions and abbreviations

3.1 Definitions

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

Advanced Electronic Signature (AdES): advanced electronic signature means an electronic signature that meets the following requirements [i.15]:

1) it is uniquely linked to the signatory;
2) it is capable of identifying the signatory;
3) it is created using means that the signatory can maintain under his sole control; and
4) it is linked to the data to which it relates in such a manner that any subsequent change of the data is detectable.

NOTE: In the rest of the present document the term "signature" is used to denote an Advanced Electronic Signature.

Certificate: an electronic attestation that links a signature verification data to a person, and confirms the identity of that person [i.15]

Certificate Identifier – an unambiguous identifier of a Certificate

Certificate path (chain) validation: process of checking that a certificate path (chain) is valid

Certification-Service-Provider (CSP) – an entity or a legal or natural person who issues certificates or other services related to electronic signatures [i.15]

Certificate validation: process of checking that a certificate or certificate path is valid

Commitment Type: a signer-selected indication of the exact implication of an electronic signature

Constraints: abstract formulation of rules, values, ranges and computation results that a **Signature,** as defined above, can be validated against

Cryptographic Token: a personal security device capable of performing cryptographic operations.

Data to be signed (DTBS): SDR together with any signature attributes that are bound together with the document by the signature

NOTE: Data To Be Signed is one of the actual inputs to the cryptographic signing algorithm. The specific way that such data is provided as input is defined in the specification for the signature type used.

Data to be Signed Formatted (DTBSF): the components of the Data to be signed which have been formatted and placed in the correct sequence for signing according to the requirements of the signed data object type selected by the signer;

DTBS-Representation (DTBSR): data sent by the Signature Creation Application to the Signature Creation Device for signing;

Driving Application (DA): application that calls the SVA in order to validate electronic signatures

NOTE: The SVA returns the validation result to the DA.

Electronic Signature: data in electronic form attached to, or logically associated with other electronic data and which serves as a method of authentication of that data [i.15]

Long Term Validation (LTV): ability to validate signatures many years after the signing took place, even if e.g. certificates used in the signature have expired or revoked or algorithms used have been broken

Object Identifier (OID): a sequence of numbers that uniquely and permanently references an object

Qualified Certificate (QC): a certificate which meets the requirements laid down in Annex I of the Directive [i.15]and is provided by a certification-service-provider who fulfils the requirements laid down in Annex II of that Directive

Qualified Electronic Signature (QES): an advanced electronic signature which is based on a qualified certificate and which is created by a secure signature creation device (Note: definition based on art. 5.1 of the Directive [i.15]

Proof Of Existence (POE): evidence that proves that an object (a certificate, a CRL, signature value, hash value, etc.) existed at a specific date/time, which may be a date/time in the past

Secure Signature Creation Device (SSCD): a signature creation device that meets the requirements laid down in Annex III of the EU Directive [i.15]

Signatory: a person who holds a signature creation device and acts either on his own behalf or on behalf of the natural or legal person he represents. Note: The term 'signer' is used throughout this document as a synonym [9]

Signer: see Signatory

Signature Attributes: see signature properties

Signature Creation Application (SCA): the application within the SCS that creates an electronic signature, excluding the SSCD/SCDev;

Signature Creation Data (SCD): unique data, such as codes or private cryptographic keys, which are used by the signatory to create an electronic signature [i.15];

Signature Creation Device (SCDev): configured software or hardware used to implement the signature-creation data; [i.15]

Signature Creation Environment (SCE): the physical, geographical and computational environment of the signature creation system;

Signature Creation Policy: set of rules for the creation of an electronic signature, under which the signature can be determined to be valid

Signature Creation System (SCS): the overall system, consisting of the SCA and the SSCD/SCDev, that creates an electronic signature;

Signature Invocation: a non-trivial interaction between the signer and the SCA or SSCD/SCdev that is necessary to invoke the start of the signing process in the SCA/SSCD to generate the Signed Data Object. It is the 'Wilful Act' of the signer.

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 Properties – Additional information that is signed together with the SDR; also called *signature attributes*

Signature Suite: combination of a cryptographic signature algorithm with its parameters, a key generation algorithm, a padding method, and a cryptographic hash function (ETSI SR 002 176 [7])

Signature type: specific format for encoding an advanced electronic signature including its attributes

Signature Upgrade: the process by which certain material (e.g. time-stamps, validation data and even archival-related material) is incorporated to an existing electronic signature aiming at making them more resilient to change or enlarging their longevity

Signature validation: process of checking that a signature is valid including overall checks of the signature against local or shared signature policy requirements as well as certificate validation and signature verification

Signature Validation Application (SVA): application that implements the signature validation processes defined in the present document

NOTE: The Signature Validation Application takes inputs from and provides validation results to a Driving Application (DA).

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

Signature verification: process of checking the cryptographic value of a signature using signature verification data

Signature Process Result Object (SPRO) or Signature Process Output: this contains the result of the SCA signature process consisting of the digital signature over the DTBS, an SDR as well as Signature Attributes. It is in a format specified by the signer selected Signed Data Object Type;

Signed Data Object Type: the type of the Signed Data Object (e.g. as specified in [1, 2, 12-15]), which specifies the resultant content and format of the SDO that is output from the SCA;

Signer's Authentication Data: data (e.g. PIN, password or biometric data) used to authenticate the signer to the SCDev and which is required to allow the use of the signature creation data held on the SCDev. The signer's authentication data may be referred to as 'Activation Data' in other documents;

Signer's / Signers' Document (SD): the data for which one or more signers intend to create an Electronic Signature or for which an Electronic Signature was created;

NOTE: In some formats it is possible to sign parts of documents only. In such a case the SD is that part of the original document that is intended to be signed. The parts that are excluded from the signature are irrelevant for the considerations made here.

Signer's / Signers' Document Representation (SDR): an element representing the SD for inclusion into the signature; e.g. a hash of the SD or some element including the hash of the SD or, eventually, the SD itself.

Trusted Path: A path between two entities or components within an SCA that provides integrity, authenticity and confidentiality;

Validation constraint: criterion, applied by an SVA when validating an electronic signature

NOTE: Validation constraints may be defined in a formal signature policy, may be given in configuration parameter files or implied by the behaviour of the SVA.

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

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

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

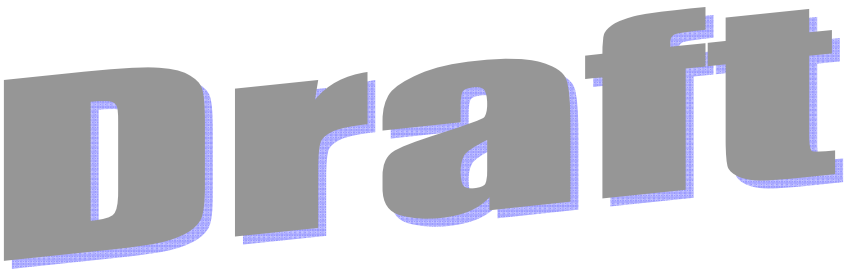
3.2 Abbreviations

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

AA	...
AC	...
AdES	Advanced Electronic Signature
$AdES_{QC}$	Advanced Electronic Signature supported by a Qualified Certificate
BES	Basic Electronic Signature
CA	Certification Authority
CAdES	CMS Advanced Electronic Signatures

CD	Commission Decision
CEN	Comité Européen de Normalisation (European Committee for Standardization)
CRL	Certificate Revocation List
CSP	Certification Service Provider
CV	Cryptographic Verification
DA	Driving Application
DHC	Data Hashing Component
DN	Distinguished Name
DTBS	Data to be Signed
DTBSR	Data To Be Signed Representation
DTBSF	Data To Be Signed Formatter
EC	European Commission
EPES	Explicit Policy-based Electronic Signature
ERS	Evidence Record Syntax
IP	Internet Protcol
ISC	Identification of the Signer's Certificate
LCP	Lightweight Certificate Policy
LDAP	Lightweight Directory Access Protocol
LT	Long Term
LTA	Long-Term with Archive Time Stamp
LTV	Long Term Validation
NCP	Normalized Certificate Policy
NO_POE	NO Proof Of Existence
OCSP	Online Certificate Status Provider
OID	Object Identifier
PIN	Personal Identification Number
PAdES	PDF Advanced Electronic Signatures
PKIX	Public Key Infrastructure X. 509
POE	Proof Of Existence
QC	Qualified Certificate
QES	Qualified Electronic Signature
QCP	Qualified Certificate Policy
RFC	Request For Comment
RSA	Rivest-Shamir-Adleman
SAC	Signer's Authentication Component
SAD	Signer's Authentication Data
SAV	Signature Acceptance Validation
SCD	Signature Creation Data
SCE	Signature Creation Environment
SCA	Signature Creation Application
SCS	Signature Creation System
SD	Signers' Document
SDC	SD Composer
SDO	Signed Data Object
SDOC	Signed Data Object Composer
SDP	SD Presenter
SDR	Signers' Document Representation
SHI	SCDev Holder Indicator
SIC	Signer's Interaction Component
SLC	Signature Logging Component
SPV	Signature Property Viewer
SSC	SCDev/SCA Communicator
SSCD	Secure Signature Creation Device
ST	Short-Term
SVA	Signature Validation Application
TA	Trust Anchor
TSA	Time-Stamping Authority
TSL	Trust-service Status List
TST	Time-Stamp Token
URI	Uniform Resource Identifier
VCI	Validation Context Initialisation
XAdES	XML Advanced Electronic Signatures

XCV	X.509 Certificate Validation
XL	Extended Long electronic signature
XML	Extendable Mark-up Language



4 Signature Creation

4.1 Lifecycle of an electronic signature

4.1.1 Introduction

Figure 1 illustrates the potential life cycle an advanced electronic signature can potentially go through. Note that most signatures created will only encounter some of the steps in the life cycle. This section will describe each step in the life cycle.

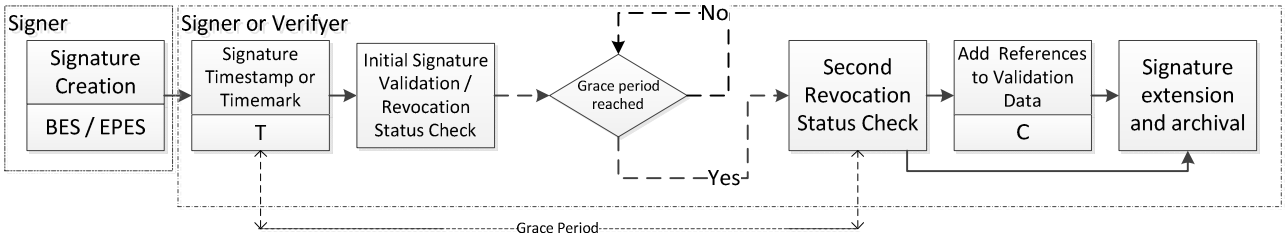


Figure 1: Signature Lifecycle

Each of the steps in the life cycle corresponds to an electronic signature form. AdES format specifications specify the implementation of these forms in specific formats (ADD Ref). This clause is applicable to all implementations of advanced electronic signatures, irrespective of the format used.

4.1.1.1 Grace Period

A grace period permits certificate revocation information to propagate through the revocation processes. This period could extend from the time an authorized entity requests certificate revocation, to when relying parties may be expected to have access to such revocation information. This typically means the issuance of a new CRL or the availability of the new certificate status to the OCSP responder. In order to make sure that the certificate was not revoked at the time the signature was time-marked or time-stamped, a signature validation policy MAY force verifiers to wait until the end of the grace period. An illustration of a grace period is provided in .

Note: In many scenarios, waiting for an extended time until accepting a signature will be incompatible to standard business requirements. The validation policy used should reflect such requirements.

4.1.2 Initial Signature Creation

Advanced electronic signatures conforming to [1,2,12] build on a base format (e.g. 10, 16) by incorporating qualified properties into the signature. Some of the properties will be covered by the signer's signature (signed qualifying information) while others will not (unsigned qualifying information).

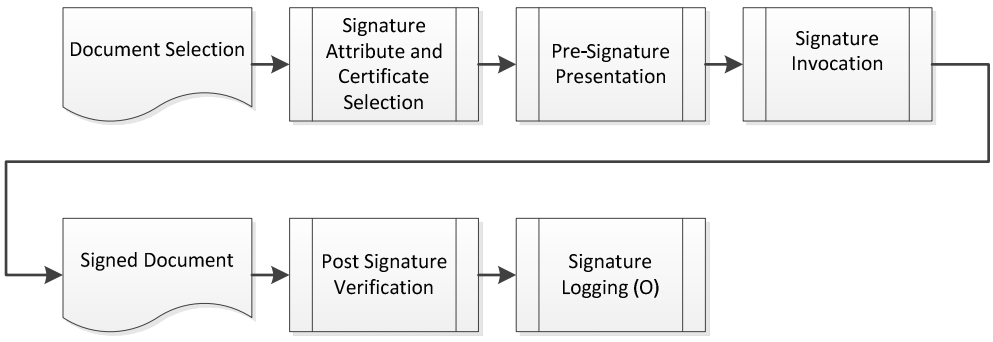


Figure 2: Initial Signature Creation

shows the steps involved in creation a signature. Clauses 4.1.2.3.1 to 4.1.2.3.4 describe these steps while clauses 4.1.2.2.1and 4.1.2.2.2describe the two possible forms of Advanced Electronic Signatures resulting from this process.

4.1.2.1 Inputs

Table 1: Inputs to the Initial Signature Creation process

Input	Requirement
Document or Document Hash	Mandatory
Signer's Certificate	Mandatory
Signer's Authentication Data	Mandatory
Signature Policy	Optional
SD Data Content Type	Optional
Commitment Type	Optional
Other Signature Attributes	Optional

4.1.2.2 Outputs

4.1.2.2.1 Basic Advanced Electronic Signature (AdES-BES)

A **Basic Advanced Electronic Signature** (BES) SHALL contain a reference to the signer's certificate as a signed qualifying property; they are designed to prevent simple substitution and reissue attacks and to allow for a restricted set of certificates to be used in verifying a signature.

NOTE 1: Additional mandatory attributes may be format specifically defined.

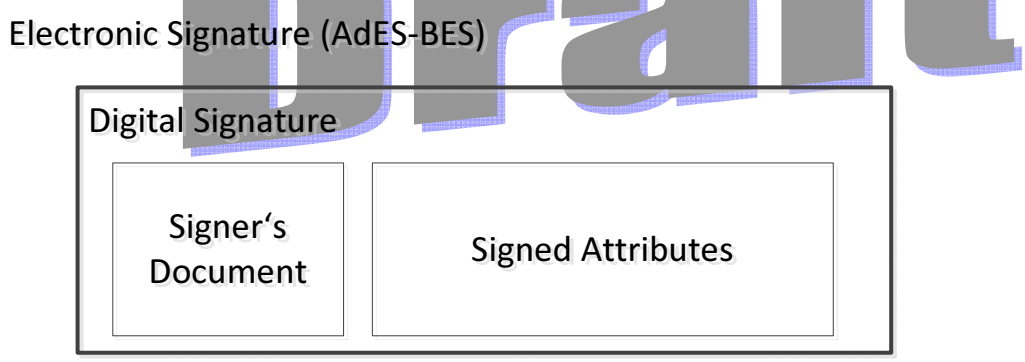


Figure 3: AdES-BES

4.1.2.2.2 Explicit Policy-based Electronic Signature (AdES-EPES)

An **Explicit Policy-based Electronic Signature** (EPES) extends the definition of an electronic signature to conform to an identified signature policy. It incorporates a signed attribute indicating the signature policy that is recommended to being used to validate the signature.

Note: TODO Add Note regarding MUST or Recommended – after discussion

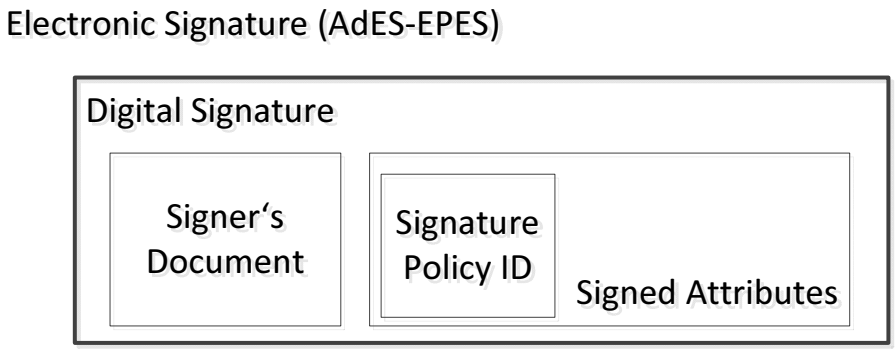


Figure 4: AdES-EPES

4.1.2.3 Processing

4.1.2.3.1 Document Selection

The signer uses a document composer (SDC) to create or select a document that is going to be signed. In the selection process, the signer may have the possibility to select the hash of a document instead.

4.1.2.3.2 Signature Attribute and Certificate Selection

This step allows the signer to select the Certificate that is appropriate for the type of signature required as well as other signature attributes (see clause 4.5.2). The signer also can select the required Signed Data Object Type to specify the required form and content of the result (SDO).

4.1.2.3.3 Pre-Signature Presentation

The SD Presentation Component (SDP) is intended to ensure that the SD and the intent of the signature is unambiguous by presenting the Signers' Document and the Signature Attributes so that they can be inspected. It should be possible for the signer to examine all Signature Attributes, but in particular the signer must be able to check the content of the following:

1. The Signer's Certificate
2. The SD Data Content Type (if present);
3. The Signature Policy (if present);
4. The Commitment Type (if present);

Use of a revoked or expired certificate can lead to creation of invalid signatures. An SCA should therefore check the validity period of the signer's certificates before signing. The SCA should also check the revocation status of the certificate. This can be achieved, for example, either by accessing the CSP's Certificate Revocation Lists, or by accessing an appropriate Online Certificate Status Provider service.

4.1.2.3.4 Signature Invocation

According to its definition, an advanced electronic signature is uniquely linked to the signer (see). Technically, this is achieved in two steps: A link between the signer and the signature creation device (Unique link 1 in the figure) and a link between the signature creation device and the signature (Unique link 2).

Unique link 1 means technically that the Advanced Electronic signature can only have been created by an SCDev with the related signature creation data corresponding to the signature verification data from a qualified certificate. Unique link 2 means technically that the SCDev has to verify that the legitimate signer is the one who requires a signature creation. If there are other means for keeping the SCDev under the sole control of the signer, then they are also applicable.

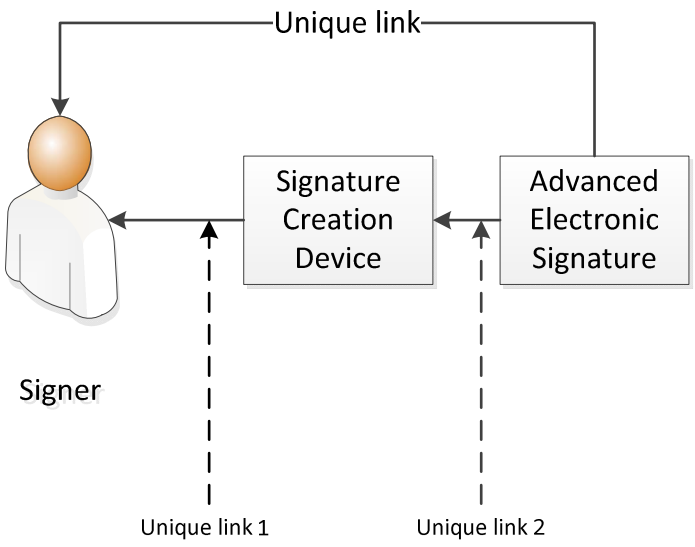


Figure 5: Unique Link

Prior to creating a signature, the SCA must determine that the signer really wants to create an Advanced or, where applicable, Qualified Electronic Signature, and that this cannot come about by accident. This can be achieved by the SCA (or SCDev) prompting the signer to commit a sequence of pre-specified non-trivial interactions over the SIC. In this document this is referred to as the 'Signature Invocation'.

A Signature Invocation is a signal from the signer to the SCA over the SIC component indicating that the signer is satisfied that the SCA is referencing the correct SD and the correct Signature Attributes as verified by the presentation processes, and that the signer wishes to create an Advanced or, where applicable, Qualified Electronic Signature covering them, i.e. to sign the document.

The SCA must ensure that each signature generated is the result of an explicit Signature Invocation.

4.1.2.3.4.1 Signer Authentication

To ensure the unique link between the electronic signature and the signer, the SCDev performs an authentication procedure to verify that the legitimate SCDev holder is the one requesting creation of an electronic signature. Two types of signer authentication are possible:

- knowledge based signer authentication (i.e.: based on a PIN or password);
- biometric signer authentication.

After the signer has successfully presented the Signer's Authentication Data (e.g. a PIN, a password, a fingerprint), the "Security Status" of the SCDev is set to allow signing. Whether this security status is maintained, (i.e. the SCDev allows the creation of several signatures), or not (i.e. Signer Authentication is required for each signature created), depends on the signing application options and on the security policy defined by the user or user's organisation. For instance, a doctor signing many (150) prescriptions would not really need to perform 150 full signature invocations in a single session, but all 150 signatures must still be considered to have been generated as a single wilful act. However, an executive signing high value contracts may want to place a limit of one signature for each invocation.

4.1.2.3.4.1.1 Obtaining the Signer's Authentication Data

Before creating an Advanced, Electronic Signature, the SCDev (and possibly the SCA) must make sure that the signer is the owner of (or is authorised to use) the SCDev. It does this by obtaining the Signer's Authentication Data from the signer. In some SCA/SCDev configurations, the Signer's Authentication Data is passed from the signer through the SCA, and then transferred to the SCDev.

4.1.2.3.4.1.2 Knowledge based Signer Authentication

In a knowledge based authentication process, the Signer presents a secret to the SCA or SCDev. Examples of such secrets are

- a Personal Identification Number (PIN); or
- a Password (PW).

This secret is referred to as knowledge based Signer's Authentication Data. The SCDev compares this against a stored reference data copy of it held by the SCDev and produces a positive verification result if they match.

4.1.2.3.4.1.3 Biometric Signer Authentication

In biometric based authentication, the signer presents a biometric feature from which the biometric signer's authentication data is derived.

For some biometric systems, like those based on fingerprints, extraction of the template is done inside the biometric terminal. The template is captured at registration time (enrolment), logically linked to the user, and kept either in a central data base or in a hardware token (the SCDev) carried by the user and presented at authentication time.

4.1.2.3.4.2 Post Signature Verification

Despite all of the security measures implemented in the SCA, some form of corruption or substitution of one or more DTBS components or DTBSF may take place. Therefore it is strongly recommended that signers are provided with the facilities to enable them to check that a verifiable electronic signature has been applied to the correct Signer's Data and Signature Attributes – i.e. as a check on the overall SCA and SCE's correct functionality.

4.1.2.3.4.3 Signature Logging (Optional)

Signature Logging is optional and also discussed in 4.3.6.8.

4.1.3 Electronic signature with time (AdES-T)

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

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

Figure 6: AdES-T

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

Trusted time provides the initial steps towards providing long term validity. The AdES-T trusted time indications need to be created before a certificate has been revoked or expired. If this cannot be achieved, the created signature cannot be validated.

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

4.1.4 Initial Signature Validation

After having created an electronic signature successfully, it is good practice, and thus recommended, to perform an initial signature validation. If such an initial validation is successful, a relying party should be able to validate the signature as well, provided that she follows an equivalent validation policy. This initial validation shall conform to the process described in clause 5.3.

Note 1: During this initial validation certificates and revocation information required to do a complete validation will automatically be collected and can be used for creating other forms of advanced electronic signatures (AdES-C and AdES-X-L).

Note 2: Initial Signature Validation may fail with a reason-code TRY_LATER, e.g. if the revocation information is not fresh enough to ensure the signers certificate is valid. Such failures would not mean that the signature creation had failed.

4.1.5 Electronic signature with complete validation data references (AdES-C)

Advanced Electronic Signatures with Complete validation data references (AdES-C) in accordance with the present document adds to the AdES-T unsigned properties containing references to the full set of CA certificates that have been used to validate the electronic signature up to (but not including) the signing certificate as well as a full set of references to the revocation data that have been used in the validation of the signer and CA certificates.

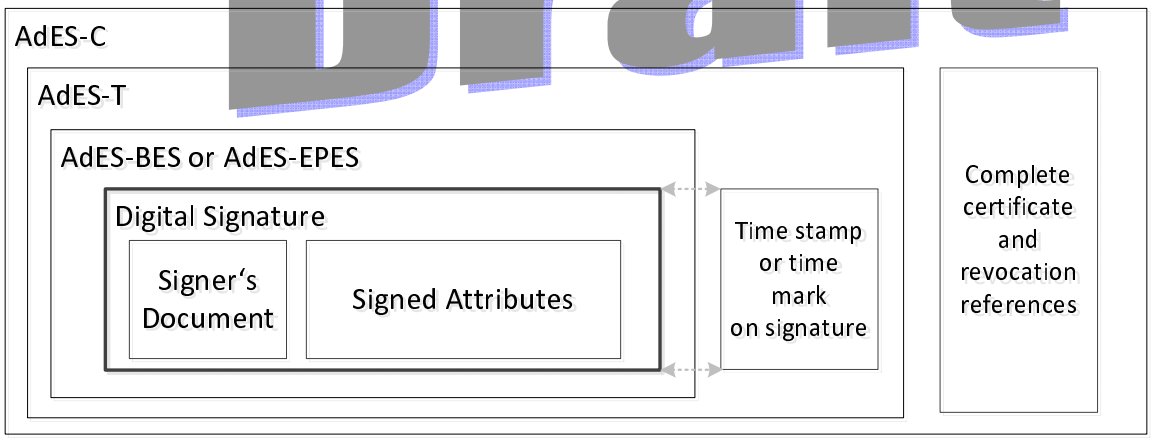


Figure 7: AdES-C

If attribute certificates appear in the signature, then AdES-C also incorporates references to the full set of Attribute Authorities certificates and references to the full set of revocation data that have been used in the validation of the attribute certificates present in the signature, respectively.

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

4.1.6 Extended electronic signature forms

The AdES forms specified in clause 4.1.2.2.1, 4.1.2.2.2 and 4.1.3 can be extended by adding certain unsigned properties that are defined in the clauses below. These properties are applicable for very long term verification and are intended for proper handling of certain disaster situations such as key-compromise or broken algorithms.

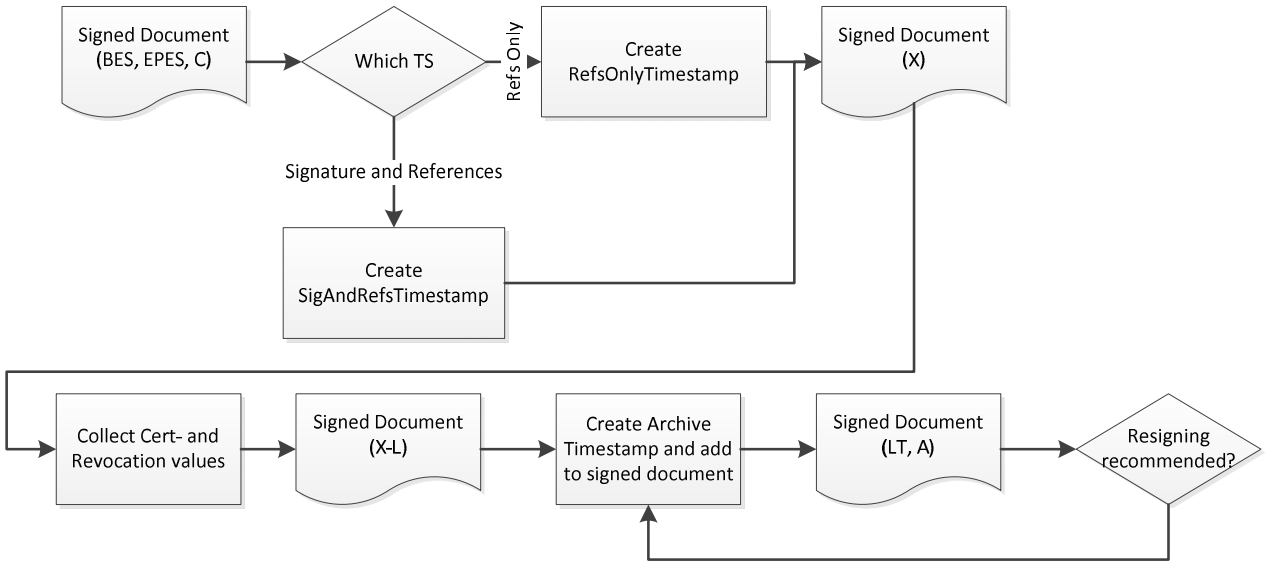


Figure 8: Extending signatures

Extending signatures will typically be done by the signer or a verifier, but can be done by any party that has interest in preserving the possibility to validate the signature in the future. Figure 8 shows the typical sequence of extensions: Starting with an AdES-C, the certificate and revocation references are secured using time-stamps (AdES-X). Then, in addition to references validation data is added to the signature (AdES-X-L). Time stamping the whole package secures the collected information (AdES-A) and can be repeatedly applied. The clauses below give details of these forms of extended signature formats.

4.1.6.1 Extended signatures with time indication (AdES-X)

Extended signatures with time indication forms (AdES-X) build on signatures containing complete certificate references and complete revocation references (AdES-C), by adding one or more time-stamps.

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

4.1.6.1.1 AdES-X type 1

AdES-X type 1 is built by adding one or more time-stamps (obtained from different TSAs). These time-stamps are computed on the signature value element, the signature time-stamp if present, and the set of complete certificate and revocation references.

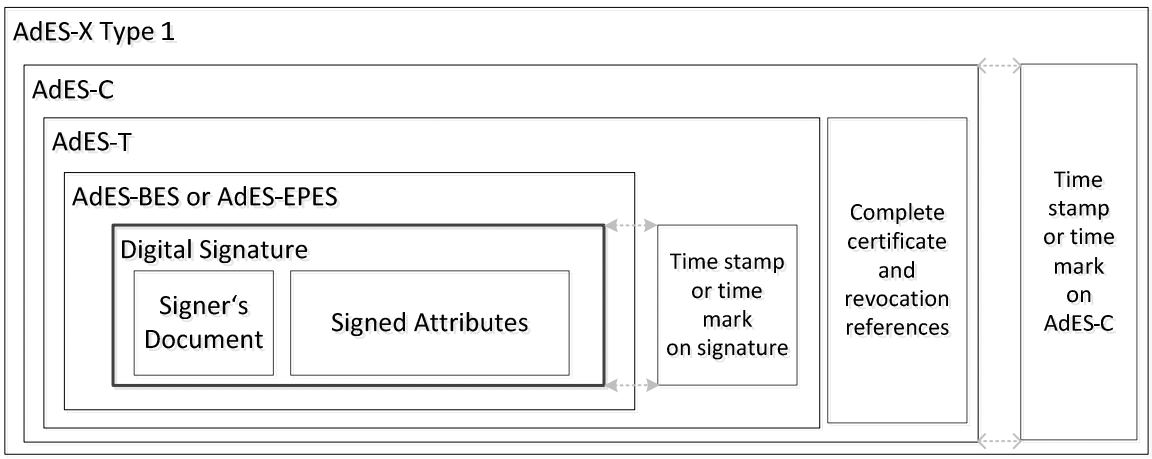


Figure 9: AdES-X- Type 1

4.1.6.1.2 AdES-X type 2

AdES-X type 2 is built by adding one or more time-stamps (obtained from different TSAs). These time-stamps are computed only on the set of complete certificate and revocation references.

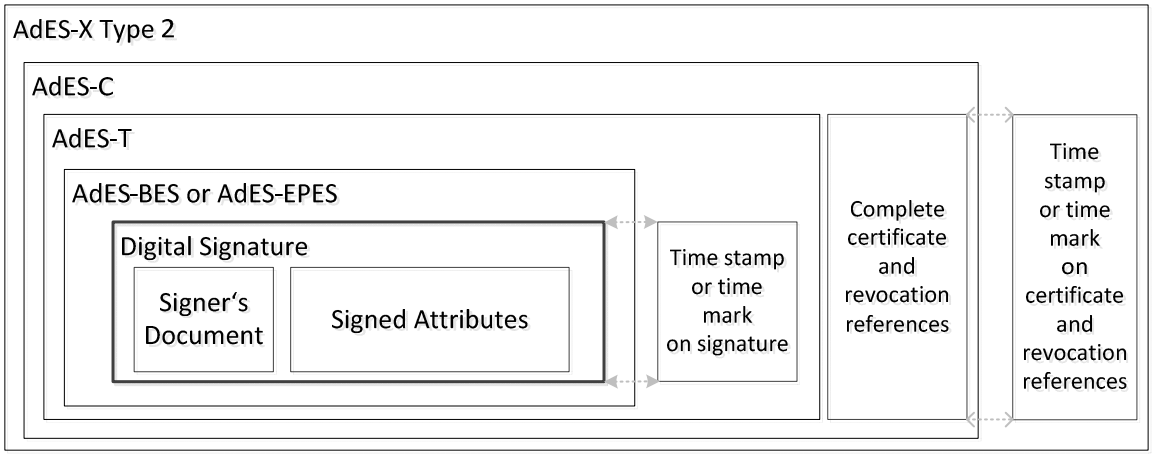


Figure 10: AdES X Type 2

4.1.6.2 Extended long signatures with time indication (AdES-X-L)

Extended long electronic signatures with time (AdES-X-L) forms in accordance with the present document build up on AdES-X types 1 or 2 by adding certificate and revocation values.

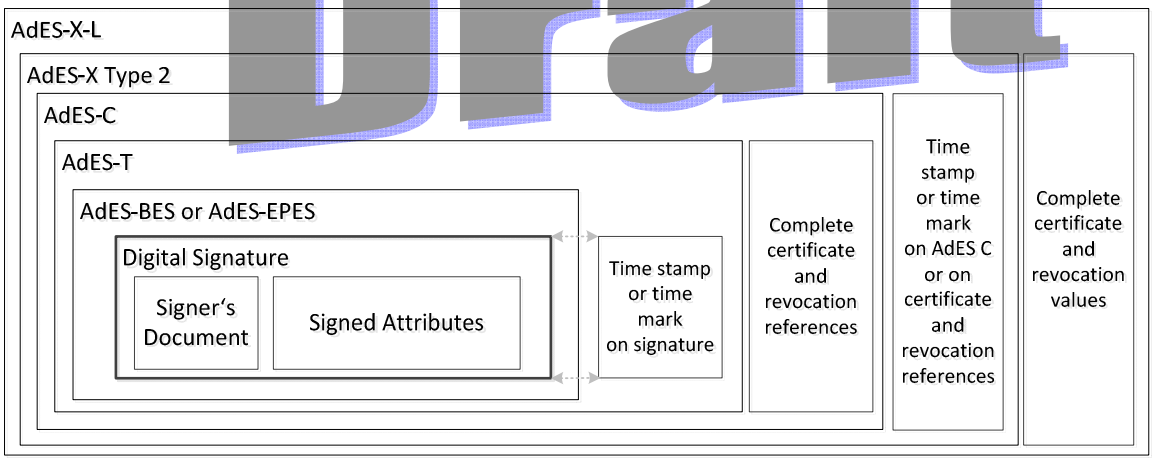


Figure 11: AdES X-L

This form of electronic signature can be useful when the verifier does not have direct access to the following information:

- 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 AdES-C.

4.1.6.3 Archive Validation Data (AdES-A)

Before algorithms, keys, and other cryptographic data used at the time an AdES-C was built become weak and the cryptographic functions become vulnerable, or the certificates supporting previous time-stamps expire or are revoked, the signed data, the AdES-C, and any additional information (i.e. any AdES-X) should be time-stamped. If possible, this should use stronger algorithms (or longer key lengths) than in an original time-stamp. This additional data and time-stamp is called archive validation data required for the ES Archive format (AdES-A). The time-stamping process may be repeated every time the protection used to time-stamp a previous AdES-A becomes weak. An AdES-A may thus bear multiple embedded time-stamps.

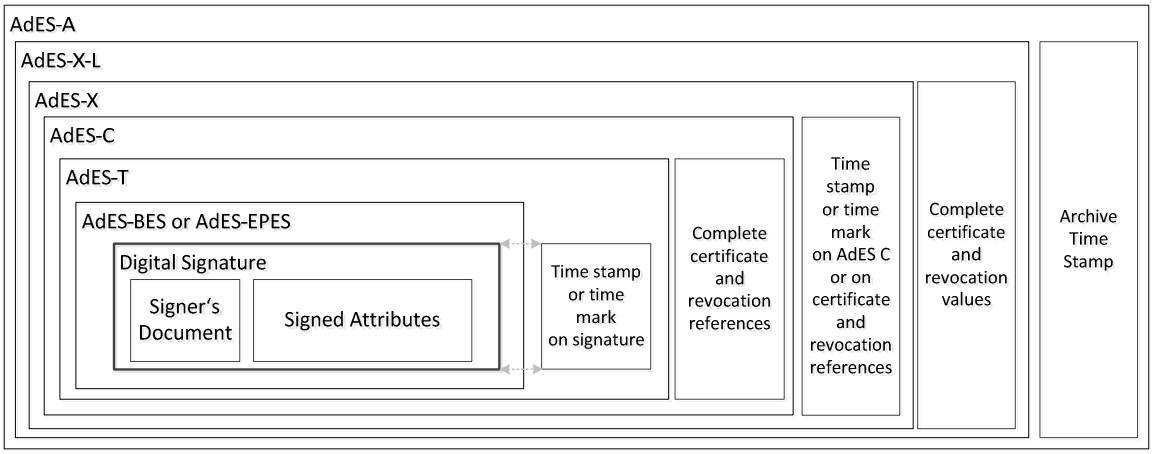


Figure 12: AdES-A

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

4.1.6.4 Long Term Validation Data (AdES-LT)

As long as a validation algorithm can assess the validity of the electronic signature, the AdES-T, or any subsequent form can be completed with a Long-Term Validation attribute.

This format is similar in spirit to the AdES-XL and AdES-A form. The core differences are:

- AdES-LT can be built upon any format above AdES-T.
- There is no restriction on the data (certificates, revocation material) that can be placed within the long-term-validation attribute.

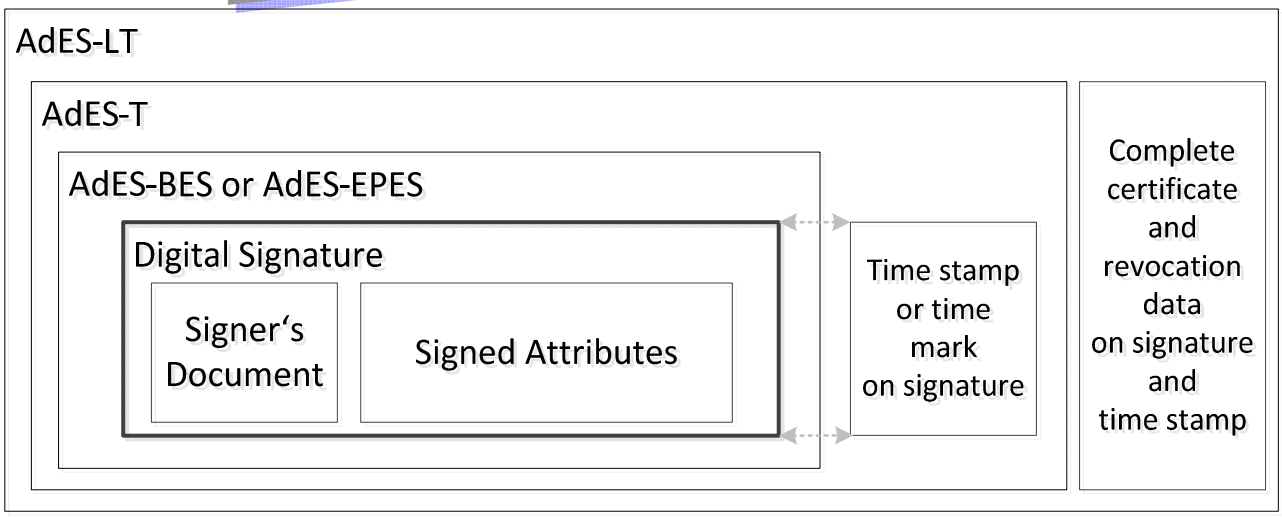


Figure 13: AdES-LT

In an AdES-LT, it is intended that the set of certificates and revocation material be sufficient to ascertain the validation status of all end-entity certificates (signer certificate, timestamps certificates, attribute certificates, ...) contained in the electronic signature. There may be more elements than necessary and may also be less elements than necessary if it is expected that recipients have an alternate means of obtaining relevant proofs of existence on these elements.

When adding a long-term-validation attribute, it is expected that the electronic signature in its current state is fully verified and that all material used during validation but not already present in the signature is added.

Similarly, when this attribute is added to a list of unsigned attributes which already contains one or several instances of this attribute, extra validation objects are gathered, which are needed to validate at the current date the proof of

existence (PoE) included in the most recent existing long-term-validation instance. "Extra objects" include the certificates and revocation objects which are not already present in that previously issued PoE.

4.1.7 Multiple Signatures

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

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

- independent signatures;
- embedded signatures
- countersignatures

Independent signatures are parallel signatures where the ordering of the signatures is not important.

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

Countersignatures are special forms of embedded signatures. Each additional signature may sign in turn the latest previously generated signature, or all the previously generated signatures and the signed document. Such subsequent signatures may be stored as a property of the countersigned signature.

4.1.8 Arbitration

In case of arbitration, a form conformant to the –C level or higher provides reliable evidence for a valid electronic signature, provided that:

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

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

4.2 Signature Creation Objectives and Models

The overall objective of a Signature Creation Application is to generate an Advanced Electronic Signature (AdES) or, where applicable, an Advanced Electronic Signature supported by a Qualified Certificate ($AdES_{QC}$), or, where applicable, a Qualified Electronic Signature (QES) that covers the Signers' Document (SD), the signer's Certificate or, where applicable, Qualified Certificate (or a reference to it), and, conditionally, the Data Content Type of the SD.

There are a variety of ways to implement the signature creation procedures, such as

- running as (part of) an application software on a device like a PC with a graphical user interface
- as a web service
- a web application
- a command-line tool

- an integrated library or a middleware for other applications

To cope with this manifold implementation options, this specification uses a simple conceptual model by dividing software with signature creation functions into two parts:

- A signature creation system (SCS) and
- a driving application", (DA)

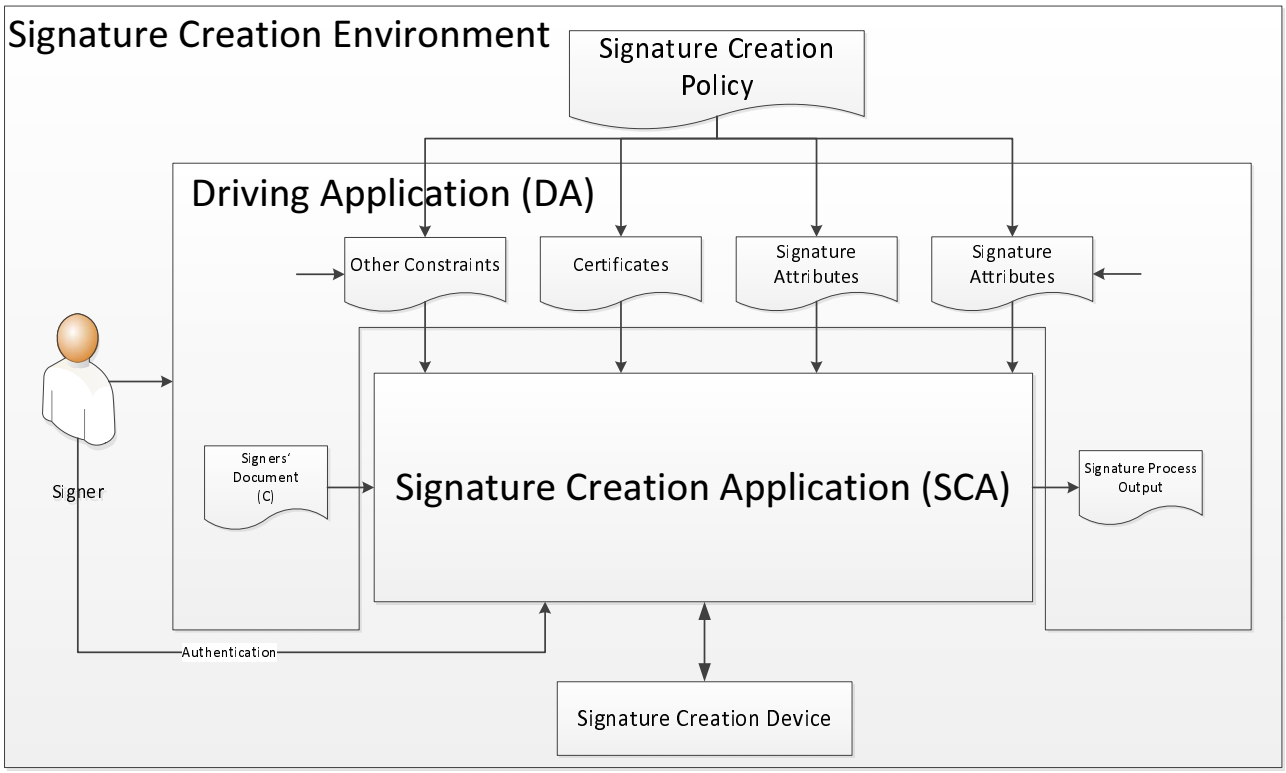


Figure 14: Conceptual Model of Signature Creation

A Signature Creation Environment (SCE) for the creation of Advanced Electronic Signatures includes a Signer interacting with a Signature Creation System (SCS) using a Driving Application (DA).

The Signature Creation System contains a Signature Creation Application (SCA), a Signature Creation Device (SCDev) or a Secure Signature Creation Device (SSCD) if a Qualified Electronic Signatures is to be created, with an associated Certificate or a Qualified Certificate if an $AdES_{QC}$ or a Qualified Electronic Signatures is to be created.

The signature creation application (SCA) receives the document to be signed together with other input from a driving application (DA), creates the electronic signature following a set of signature creation constraints and produces an output which will consist of a status indication together with the signature or signed document produced in the process.

4.2.1 Functional Model

Figure 15 shows a functional model of a Signature Creation Application (SCA) as a part of a Signature Creation System (SCS). It illustrates the signature creation functions and the information objects and interfaces that are relevant to its security. It does not distinguish between hardware or software implementations, and the model is not intended to specify the nature of any inputs/outputs or information transfer paths between the different functional components (which might take the form of direct I/O devices, hardwired connections or be distributed over communications links). Also, it makes no statement about the distribution of the functions over different platforms. These aspects can only become more concrete in the context of a particular set of technologies that apply to an SCS.

The purpose of the SCA and the SCDev is to take a SD and the related Signature Attributes, form them into Data To Be Signed (DTBS) and produce over them an Advanced, or where applicable a Qualified, Electronic Signature and to produce a Signed Data Object as a result.

The primary functions of the SCA are contained in a set of 'Trusted' and 'Application Specific' SCA components. These functions are elaborated further in this clause. In addition, the SCA will usually contain the following functions either to support the signature process or to support other functions that are not related to Electronic Signature Creation but which may have an impact on security requirements:

- An SCA Manager. This may perform a number of functions to support the signing process including the operation of the Signer's Interface, transfer of information from the Signer's Interface to the SCDev interface, interpretation of the Signature Suite and signature policy, obtaining the signature policy information and certificates, and management of local storage;
- The SCDev Interface. The SCDev is considered to be external to the SCA and will need to interact with the SCA to receive the Signer's Authentication Data and DTBS if there is no direct user interface between the SCDev and the signer, and return the digital Signature to the SCA;
- SCA local storage that may be used as a temporary location for data used during the signing process. This may also be considered as a target of security threats.

The SCA may contain other functions that are not related to signature generation e.g.:

- Data input/output ports and network connections that may be the target of security threats;
- Hardware/software processes that may also be the target of security threats;

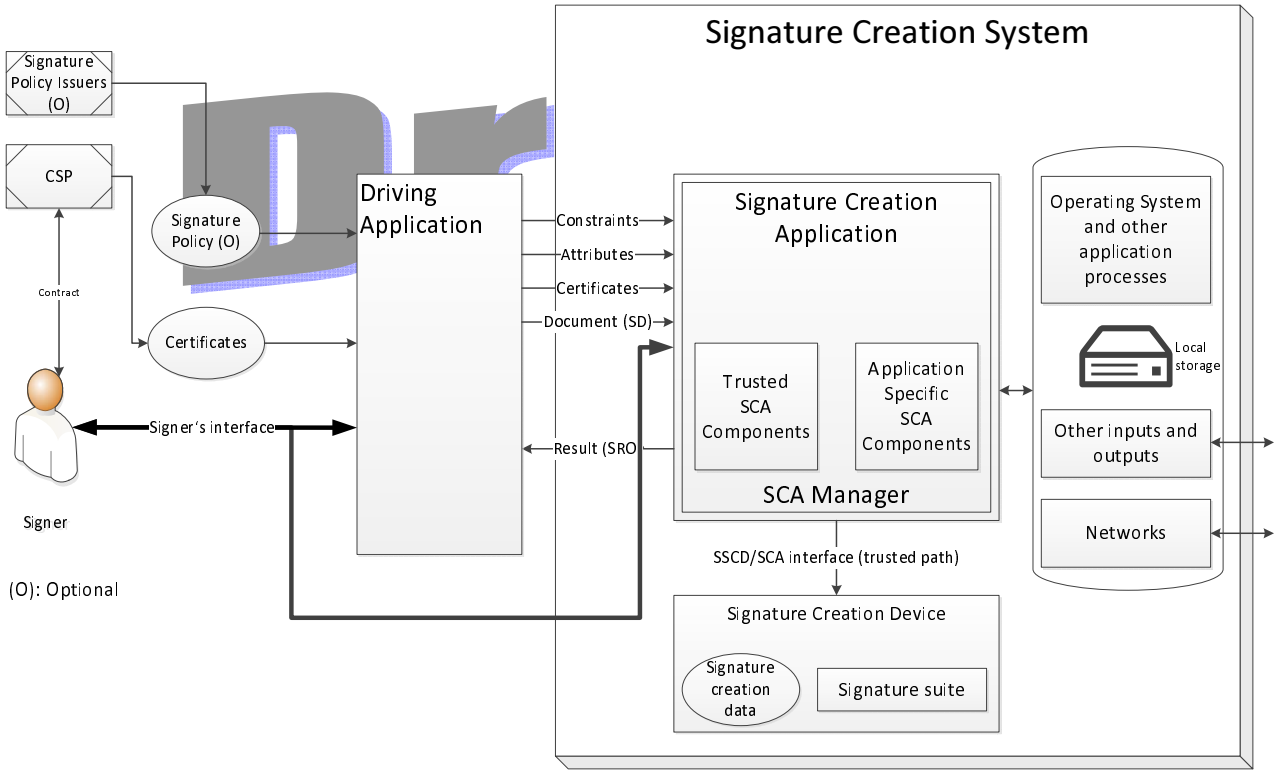


Figure 15: Signature Creation Functional Model

The following information objects, which are all detailed in clause 4.5 are used within the SCE:

- A Signature Suite;
- Signature Attributes;
- Signature Creation Data;
- Signer's Authentication Data
- Signer's Certificates;
- The Signer's Document;

The following interfaces and interactions are used to control the operation of the DA and SCA:

- Signer's interface consisting of one or more of the following
  - Selection interface for the document to be signed, allowing the signer to select the SD or the part(s) of it that has to be signed;
  - Signature Attribute selection interface allowing the signer to select the Certificate that is appropriate for the type of signature required as well as other signature attributes relevant for the signature;
  - An optional interface to the selection and inspection of signature policies (either locally predefined, configured on the fly or external) that can be used for creating the signatures.
  - Selection of the required Signed Data Object Type to specify the required form and content of the result (SDO);
  - Signer's Authentication Data Input Interface – to deliver the Signer's Authentication Data from the signer to the SCDev if the SCA is involved in this task;
- A secure presentation capability – to allow the signer to inspect the SD (or the parts of it that have to be signed), Signature Attributes and Policies prior to invoking the signature process;
- An interface to TSPs issuing certificates – over which Certificates and, optionally, Certificate Revocation Information may be obtained;
- An interface allowing configuring the SCA locally;
- An interface to other TSPs – over which e.g. time-stamping services or signature policies may be obtained
- Signature invocation – to allow the user to invoke the signing process (i.e. as a 'wilful act');
- SCDev interface – to enable the SCA and SCDev to communicate over a trusted path;
- An output of the resultant AdES as specified by the Signed Data Object Type selected by the Signer.

The specifications for the Contract between the Signer and the TSP that provides the signer with a certificate are beyond the scope of this document.

4.3 Signature Creation Application

The primary parts of a Signature Creation Application are the set of trusted and application specific components that are shown in Figure 16 and described in the following subclauses.

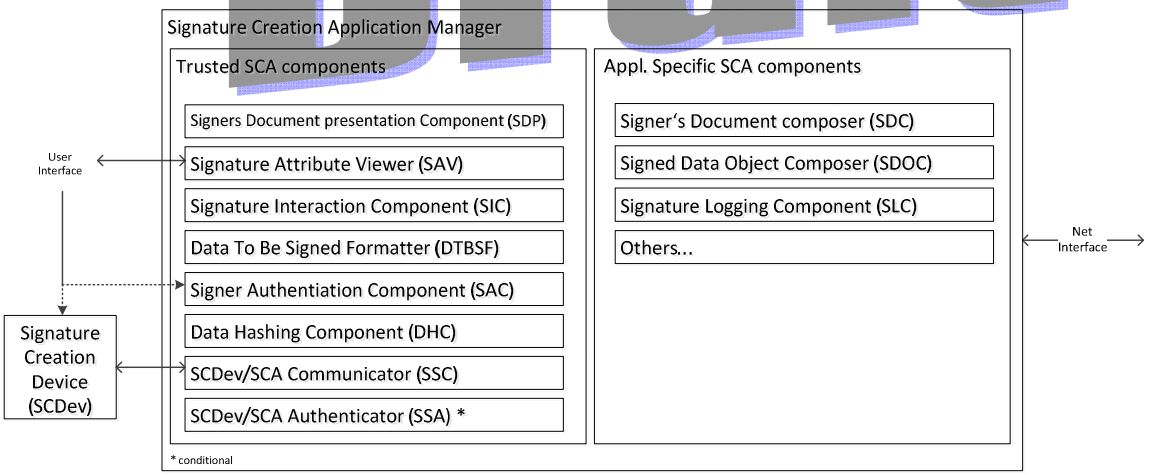


Figure 16: Signature Creation Application Components

- Each component represents a certain functionality that is required for a SCA. The functionality of a given component may or may not be implemented as a technical component in an implementation of a SCA.
- The functionalities of the trusted components are relevant for every SCA and thus are mandatory and assumed to be present in some form if not marked otherwise;

  Note: Data Hashing Component (DHC) and Signer Authentication Component (SAC) are always considered to be present in order to encourage compatibility of the SCA with the widest possible population of SCDevs.

- The application specific components are application context dependent, i.e. their presence, construction and functionality is application specific.

4.3.1 Data To Be Signed Formatter (DTBSF)

The *Data To Be Signed Formatter* formats and sequences the SD or a hash of it together with the selected Signature Attributes and delivers the result to the Data Hashing Component.

4.3.2 SD Presentation Component (SDP)

The *SD Presentation Component* is used for presenting the signers document that the signer selects by the Signer Interaction Component. It is intended to provide reasonable trust that a document or the parts of the documents that are about to be signed is/are the one that the signer intends to sign, and that it has not been, nor will be, corrupted or modified. It does this by securely presenting to the signer the document or the parts of the document about to be signed according to its Data Content Type.

A secure SD Presentation Component will be capable of presenting SDs of a limited number of Data Content Types. The SDP should issue a warning if the signer requests the SCA to sign a document of any Data Content Type that the SCA is not specifically designed to support. However, it is entirely the signer's obligation to select an appropriate Data Content Type for the SD, and to be able to determine whether the SCA complies with the requirements for the Data Content Type.

The SDP will also be able to display any signature attributes that have been selected by the signer or automatically added as a result of the signature creation policy and which are going to be covered by the signature.

4.3.3 Signer Interaction Component (SIC)

The signer interacts with the SCA using the *Signer Interaction Component* to control the signature creation process. The SCA returns error and status messages to the signer using the Signer Interaction Component. This interface is used for all interactions between the Signer and the SCA, including input/selection of the SD and Signature Attributes and Signature Policy with the exception of the Signer's Authentication Data.

4.3.4 Signer's Authentication Component (SAC)

The *Signer's Authentication Component* (e.g. a card terminal with PIN pad) is used for acquiring knowledge based Signer's Authentication Data and/or biometric features and preparation of the Signer's Authentication Data in such a way that they can be compared with Signer's Authentication Data held in the SCDev.

4.3.5 Data Hashing Component (DHC)

The *Data Hashing Component* is responsible for producing the DTBS Representation (which might be non- hashed, partially hashed or completely hashed as required by the SCDev). If the SCDev carries out all of the hash processing, then the task of this component is only to forward the unmodified DTBS Representation to the SCDev.

4.3.6 SCDev/SCA Communicator (SSC)

The SCDev/SCA Communicator manages the interaction between SCA and SCDev. Therefore it is a very sensitive component, because any malfunction (e.g. due to attacks) may result in creation of a wrong signature. Its tasks are

4.3.6.1 Establishing the Physical Communication

The SCA must have at least one physical interface suitable for communication with a SCDev. For SCDevs permanently embedded within an SCA, the availability of an appropriate interface is required, however this need not be externally accessible.

Example interfaces and related SCDevs are:

- smart cards which require a smart card interface where the card reader may e.g. beintegrated in the PC-keyboard or the system unit, or attached as a separate card terminal to a suitable PC port (serial port, parallel port, USB port), or as a PCMCIA-card reader module for laptops;
- USB tokens which require a USB interface;

- PCMCIA tokens which require a PCMCIA interface;
- other cryptographic tokens which require e.g. a slot in a system unit with an appropriate bus interface.

The interface between an SCA and an SCDev may be e.g.:

- a contact link;
- a radio link;
- an infrared link;
- a combination of links.

4.3.6.2 Retrieval of SCDev Token Information

Different types of SCDevs exist and vary e.g. in:

- the provision of signature algorithms (e.g. RSA, DSA, ECC);
- the supported key length (e.g. for RSA keys 1024 bit, 2048 bit, ...);
- the requirement for special formats of the signature input;
- the use of hash functions (e.g. none, SHA-1, RIPEMD160, SHA-256);
- the work sharing between SCA and SCDev with respect to hashing and signature input;
- formatting;
- the method and type of user authentication;
- the provision of certificates;
- the types and sequences of commands for achieving the signature creation service from the SCDev.

The presence of SCDev token information (e.g. the cryptographic token information in IC cards as defined in [22], which is compatible to the PKCS#15 [23] specification) is helpful for any SCA. However it is particularly useful for those SCAs that need to interact with different SCDev s, such as SCAs that are under control of a service provider.

To achieve this, there is a need for the SCDev to provide information that enables the Signature Creation Application to deal with their SCDev particular capabilities. This information states where data elements held by the SCDev are to be found and how they are to be used (e.g. the signature creation data, signer’s authentication data, signer’s certificates or a URL pointing to the certificates, if not stored in the SCDev).

4.3.6.3 Selection of the SCDev functionality on a multi-application platform

The SCDev functionality may be implemented on a platform (e.g. a smart card) which carries one or more SCDev functions (often referred as "SCDev Applications") and possibly other applications. Furthermore, the SCDev functions may be part of a larger application that has more functions than just the signature creation function (e.g. a home banking application). If such a multi-application platform is used as the carrier of one or more logical SCDevs, then the SCA must select one of them (e.g. by using the associated application identifier).

4.3.6.4 Retrieval of Certificates

An SCDev may carry several certificates, e.g.:

- Certificates of the Signer used for different roles, different signature algorithms etc.;
- "Attribute Certificates" (AC) of the signer, if any;
- Certificates that may be useful for a verifier to build a certification path between the signer's certificate and a root key. E.g. certificates produced by a root CA for the CA issuing the certificates of the signer or by higher level Attribute Authorities (?) (AAs) to the AAs issuing the signer’s ACs, if applicable.

The SCDev should provide the following information (e.g. in the cryptographic token information) to the SCA:

- how to retrieve the certificates;
- the reference(s) of the related signature creation data;
- which certificates belong to which chain of certificates.

If an SCA under the signer's control already has the certificates stored, then they do not have to be retrieved again, i.e. an SCA may retrieve previously relevant certificates from an SCDev and store them so that a second retrieval is not needed (this saves time).

Depending on the security policy of the issuer of the SCDev, the retrieval of all or certain certificates may be always possible or restricted, i.e. retrieval of a certificate stored in the SCDev may be possible e.g. only after signer's authentication.

Dependent on the security policy of the issuer of the SCDev or the provider of the SCDev-application, the retrieval of all or certain certificates may be allowed or restricted, i.e. retrieval of a certificate stored in the SCDev may be allowed e.g. only after signer's authentication.

If the SCDev does not contain the certificate with the signature verification data (i.e. the public key of the signer) and possibly further certificates belonging to the signer's certificate chain, then at least an unambiguous reference to the signer's certificate in the form of a Uniform Resource Locator (URL) or another form of reference (specified e.g. in the cryptographic token information) should be retrievable from the SCDev.

4.3.6.5 Selection of Signature Creation Data

If an SCDev holds more than one instance of signature creation data, then the one appropriate for the signer's intentions has to be selected. Even if the SCDev has only a single signature creation datum, it may require that a reference to it is set. To enable the selection of the correct signature creation data, the SCDev Token Information has to contain information denoting the link between a certificate (possibly selected by the signer) and the signature creation data reference. If the SCDev also requires a reference to an algorithm, then this also has to be indicated in the SCDev token information.

4.3.6.6 Performing Signer Authentication

Where applicable, i.e.: where the SCDev has no SAD input device, the SSC component receives the Signer's Authentication Data from the SAC component over a trusted path and sends it with the appropriate SCDev command to the SCDev for comparison. The result shall be:

- verification successful; or
- verification failed; or
- verification blocked due to e.g. too many consecutive faulty presentations of the Signer's Authentication Data.

The result is delivered back to the SAC component, which presents the result with an appropriate message to the signer.

4.3.6.7 Digital Signature Computation

The final step of the signature creation process is the computation of the digital signature (encryption of the DTBSR with the signer's SCD). In order to avoid usage restrictions, an SCDev should deliver a digital signature as a bit-string. The formatting of the relevant electronic signature and the results of the signing processes is context dependent and is a task of the SDOC component.

4.3.6.8 Signature Logging

If the SCA and the SCDev log completed signatures, then the relevant interactions between the SCA and the SCDev are performed after each signature creation has been completed.

4.3.7 SCDev/SCA Authenticator (SSA)

The SCDev/SCA Authenticator establishes a trusted path between SCDev and SCA. The presence of this component is conditional, i.e. it might only be present in SCAs that are under the control of public service providers and where the trusted path cannot be established by organisational means.

If the signature creation takes place at an SCA under control of a service provider (i.e. at a public SCA), then the signer needs to be able to determine whether to assume the same level of confidence as would be achieved if the SCA is under the signer's control. The confidence level for signature creation can be achieved by organisational means or by technical means.

Technical means might be:

- an SCDev authenticates the respective SCA and vice versa and
- the communication after authentication is protected by means of secure messaging, and
- the signer is able to recognise (e.g. by displaying a Signer specific display message) whether a secure interaction between the SCA and the SCDev can be assumed. The signer should be made aware that even this assumption might be insufficient in presence of malicious codes.

Note: If such an authentication procedure cannot be performed e.g. due to lack of verification keys, then this should be indicated to the signer. In any case the reliability of the authentication of an SCA by an SCDev might be affected by a malicious code that could intercept the dialogs between the SCA and the SCDev and between the SCA and the signer.

4.3.8 Work sharing between SCA and SCDev

The work in creating the DTBSR is split between the SCA and the SCDev. The way this work is shared influences the security of the implementation.

The first step after the user has invoked the signature creation process is hashing. Hashing is required since the message or document to be signed might have any length, but a signature algorithm can only process data that is less than or equal to the key length. The hash process is outside the visual and intellectual control of the user, i.e. the user is unable to calculate or recognise whether the hash value fits the DTBS or not. That means the user has to rely on this process step. The security requirement is therefore that the message delivered to the hash function is not modifiable (i.e. its integrity is protected) and that the hash function works properly.

The hash operation itself can be organised in different ways as Figure 17, Figure 18 and Figure 19 show. The second step in a signature process is padding, which formats the hash value such that it is suitable as input for the signature process and has the required security properties.

- Complete hashing including padding in the signature creation application. This may not be secure.
- Hashing in the SCA and completion of the hashing in the SCDev. This is strong.
- Hashing and completion of the hashing in the SCDev. This is even strong.

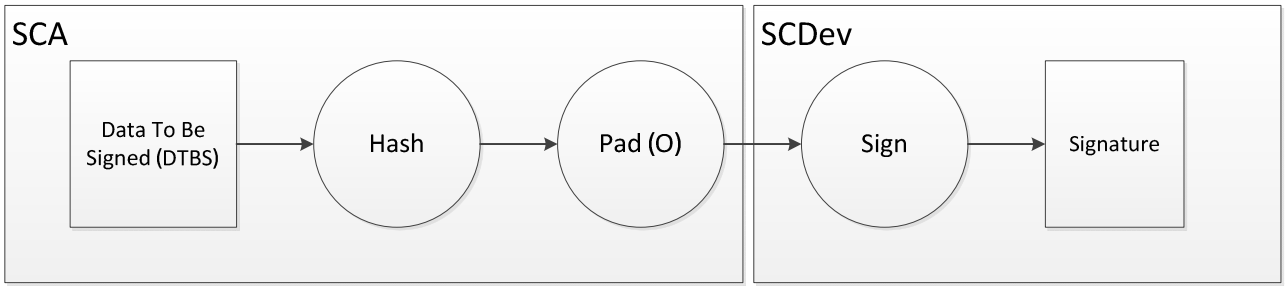


Figure 17: Hashing in the SCA

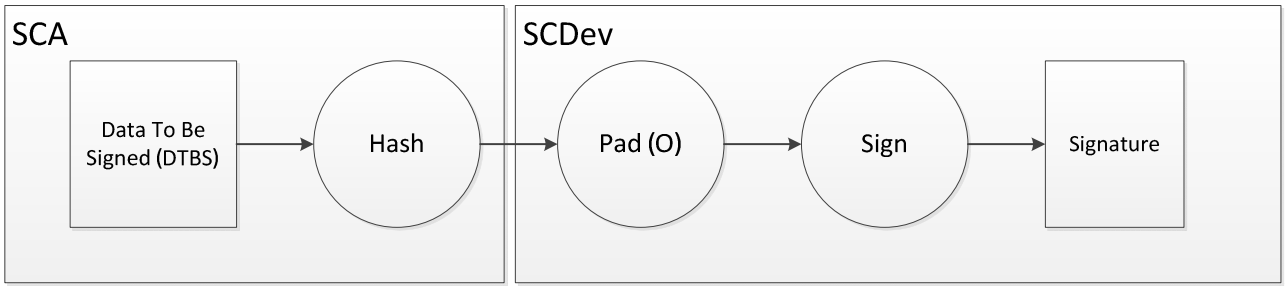


Figure 18: Partial hashing in the SCA and completion in the SCDev

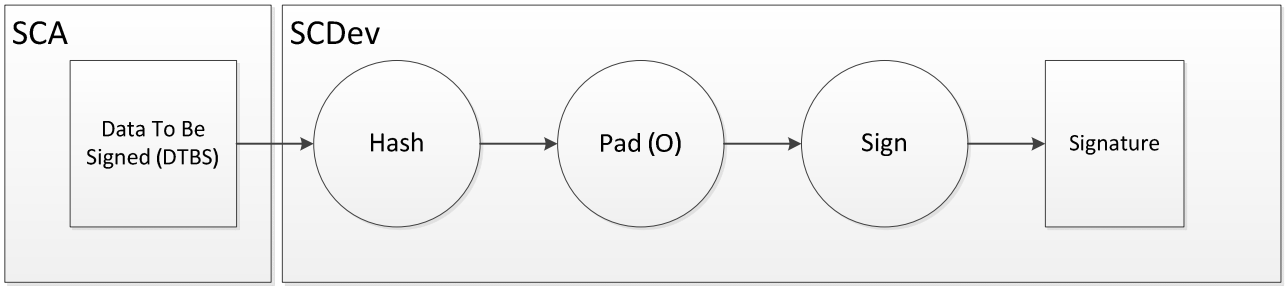


Figure 19: Complete hashing in the SCDev

Complete hashing in an SCDev of large documents is technically feasible if the SCDev has a high-speed interface like the USB SCDev.

4.3.9 Application specific components

4.3.9.1 SD Composer (SDC)

An *SD Composer* (e.g. a text editor) is used for creation, input or selection of the SD. The information that this component acts on is managed through the SIC.

4.3.9.2 Signed Data Object Composer (SDOC)

A *Signed Data Object Composer* usually takes the DTBSF components and associates them with the bit string representing the digital signature as delivered by the SCDev, and outputs the result (i.e. the SDO) of the signing process in some standard format as specified by the SDO Type (e.g. as specified in the ETSI Electronic Signature Formats [1]).

4.3.9.3 Signature Logging Component (SLC)

A *Signature Logging Component* records details of the (most recent) signatures created by the SCA.

4.4 Secure Signature Creation Devices

The SCDev performs those functions that hold the signer's signature creation data, verify the signer's authentication data and create the electronic signature using the signer's signature creation data. Examples of platforms on which SCDevs may be implemented are:

- Smart cards;
- USB Tokens;
- PCMCIA Tokens.

4.5 Signed Data Object Information Model

Figure 20 outlines and relates above mentioned building blocks and illustrates the data flow and envisioned for the process of the generation of an Electronic Signature for an SD, and illustrates its relationship to the SCA components

involved and introduced in the clauses above. The following sub-clauses describe information objects used in this process.

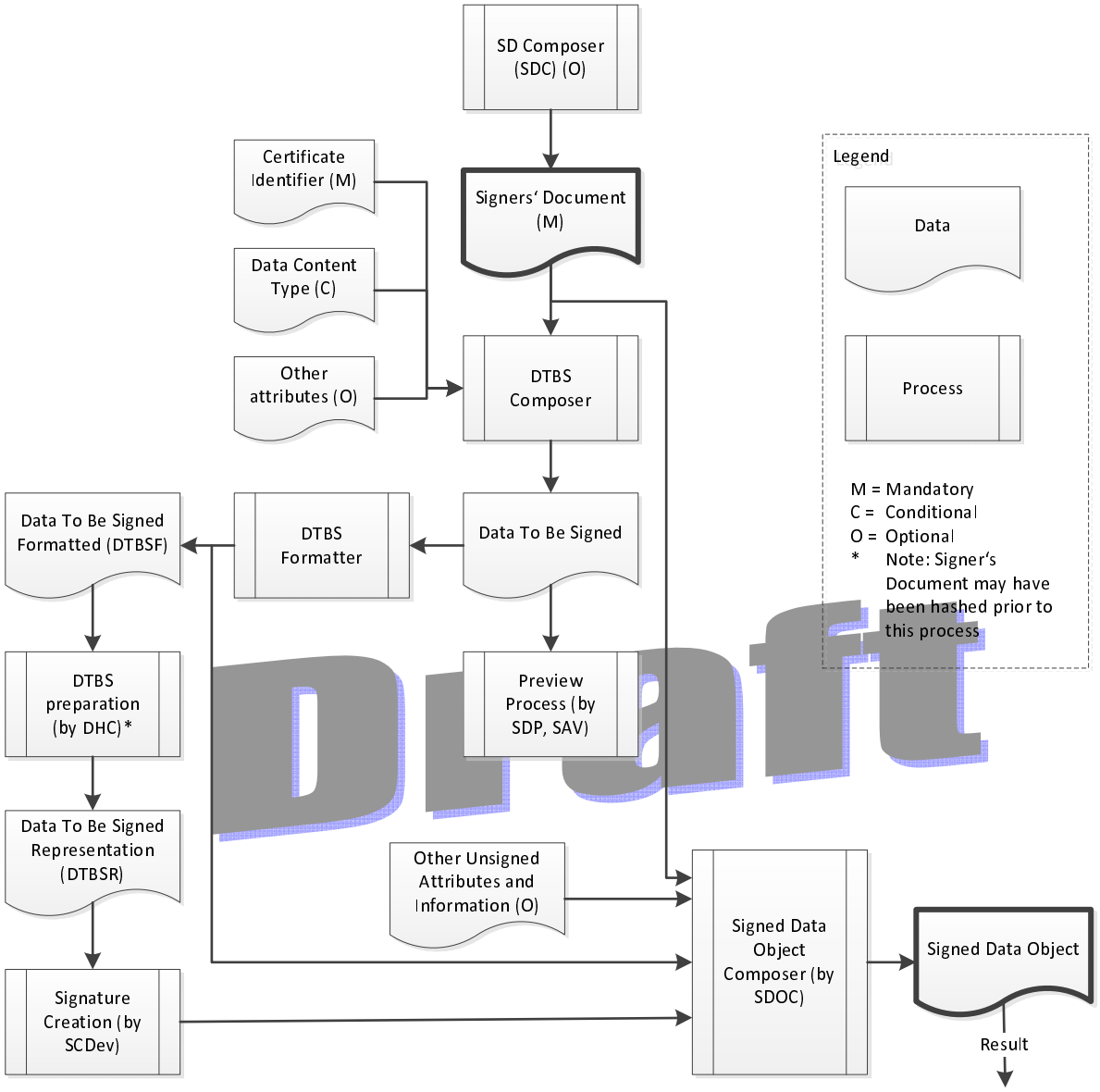


Figure 20: Information Model of Advanced or Qualified Electronic Signature Creation

4.5.1 Signer's Document (SD)

The Signer's Document (SD) is the document upon which the Advanced or, where applicable, Qualified Electronic Signature will be generated and to which it will be associated. The SD is selected or composed by the signer using the Signer's Document Composer (SDC) component. In some cases, a hash of the SD may be presented to the signature processes instead of the complete SD.

The SD potentially has a number of important variants and components that impact the signing process and the status of the signature:

1. It may be in revisable format such as a word processor document or a message or file that can be edited, and where its presentation is dependent on the current configuration of the viewing device, and where the signer can potentially be presented a representation of the SD having an appearance different from that presented to the verifier;

2. It may be in an unambiguous, un-modifiable form (e.g. txt, Postscript, ODA final form …). These formats contain complete presentation rules that guarantee that the signer and verifier can be presented the SD in the same way if the same presentation rules are followed;
3. Hidden encoded information may be present (e.g. macros, hidden text, active or calculated components, viruses …). These may not be visible to the signer during the pre-view and verification processes, and the signer may not be aware of their presence. These represent potential ambiguities in the SD and are regarded as a security threat;
4. It may be in a form that is not normally presented to the signer or verifier directly, or it may be in a form that is inherently presented to the signer and verifier in different ways (whilst representing the same semantics). Examples of these formats are Electronic Data Interchange formats, Web Pages (HTML), XML, SGML, and computer files;
5. It may potentially contain embedded Signed Data Objects that have been created by persons or entities other than the Signer. The format of the SD is described by the Data Content Type signature attribute. This attribute specifies exactly how it should be presented or interpreted by the verifier, and the type of application or presenter that a verifier should use in inspecting or using the SD.

4.5.2 Signature Attributes

Signature Attributes are pieces of information that support the electronic signature and its interpretation and purpose and which may be covered by the signature together with the SD. The Signature attributes are either input or selected by the signer through the SIC component.

This clause specifies mandatory and optional signature attributes. Attributes can either be *signed attributes*, i.e. attributes that are covered by the signature, or *unsigned attributes*, i.e. attributes that are not secured by the signature. Unsigned attributes may also be added to a signature at a later stage. The set of attributes included in a signature is defined by the signature creation policy used or, when extending a signature, by the signature validation policy used and can also be format specific.

The following subclauses specify some signature attributes that are commonly used. Further signature attributes may be specified and may be application-specific. Examples of this information and its uses are contained in the ETSI Electronic Signature Format documents [1,2,12].

4.5.2.1 Signer's Certificate Identifier

This attribute contains a copy or the identifier of, or a reference to, the certificate holding the Signature Verification Data corresponding to the Signature Creation Data that the signer uses to create the electronic signature. Its presence is mandatory if the signer's identity is not indicated by other means because the signer might hold, either at the moment of the signature or in the future, a number of different certificates that relate to the same signature creation data. This attribute prevents substitution of the referenced certificate with another one with different semantics. If the signer holds different certificates related to different signature creation data it indicates the correct signature verification data to the verifier.

This attribute may also contain references to other certificates. If so, they limit the set of certificates that are used during validation and typically form the chain for chain validation of the signers' certificate.

For each certificate, the attribute also contains a digest together with a unique identifier of the algorithm that has been used to calculate that digest.

> NOTE: This attribute is meant to identify the certificate that is to be used for verification of the signature. This makes signature validation easier. It also allows differentiating the certificate in the case where more than one certificate has been issued for one set of SCD, i.e. using one key but different certificates for different purposes. The signature would be technically verifiable using any of these certificates. The attributes in these certificates then may make a difference if the signature can be accepted by the SVA or not. While this is technically possible, it is strongly recommended to use different signature creation data for different purposes.

4.5.2.2 Signature Policy reference

A signature policy reference attribute can be present if required by the signing context (e.g. in a specified trading agreement). This reference indicates to the verifier which is the correct signature policy to be used during the verification process. For instance, a signature policy might be used to clarify the precise role and commitments that the signer intends to assume with respect to the Signer's Document.

This attribute may also contain a digest of the policy together with a unique identifier of the algorithm that has been used to calculate that digest.

4.5.2.3 Data Content Type

The Data Content Type attribute describes the format of the SD and specifies how it should be viewed and used by the verifier and as intended by the signer. It shall be included in the signature if it is not indicated by other means.

4.5.2.4 Commitment Type

This attribute contains an indication by the signer of the precise meaning of the signature in the context of the signature policy selected by the signer (i.e. where electronic signatures can express different intentions of the signer). If a signature policy reference is present, and the referenced policy lists a set of allowed commitment types, the content of this attribute shall be selected from the set specified by that policy.

4.5.2.5 Counter Signatures

This attribute contains an indication by the signer that the signature containing this attribute is a countersignature of a signature referenced in that attribute.

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

4.5.2.6 Claimed signing time

This attribute contains the time at which the signer claims to have performed the signing process.

4.5.2.7 Signed data object format

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

4.5.2.8 Indication of production place of the signature

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

4.5.2.9 Signer attributes/roles

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

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

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

4.5.3 Data To Be Signed (DTBS)

The Data To Be Signed consists of the information objects that are to be covered by the AdES or, where applicable, $AdES_{QC}$ or Qualified Electronic Signature. These are:

- the SD or the SDR;
- the Signature Attributes selected by the signer that are to be signed together with the SD.

4.5.4 Data To Be Signed (Formatted) (DTBSF)

This contains the DTBS components that have been formatted and placed in the correct sequence for the signing process by the DTBSF component. It is this information object that is covered by the digital signature as the result of the signing processes and which is included in an Advanced or, where applicable, Qualified Electronic Signature and used in verifying the signature. The format of the SDO is determined by the SDO type that has been selected by the signer. Examples of such an SDO Type are the ETSI Electronic Signature Formats [1,2,12].

4.5.5 Data To Be Signed Representation (DTBSR)

This is the result of e.g. hashing the DTBSF according to a hash algorithm specified in the Signature Suite. It is produced by the DHC component. In order for produced hash to be highly representative of the DTBSF, the hashing function must be such that it must be computationally infeasible to find collisions for the expected signature lifetime. It is to be noted that should the hash function become weak in the future, additional security measures, such as applying time-stamp Tokens, can be taken.

4.5.6 Advanced Electronic Signature (AdES)

An advanced electronic signature consists of an SDO together with either the signer's certificate holding the signer's appropriate Signature Verification Data or a reference to that certificate. It is derived from the signature input (which consists of the DTBSR and possibly some padding) using the relevant signature algorithm and the Signature Creation Data associated with the chosen certificate.

4.5.7 Advanced Electronic Signature Supported by a Qualified Certificate ($AdES_{QC}$)

An Advanced Electronic Signature Supported by a Qualified Certificate (AdESQC) is an advanced electronic signature where the signers' certificate is a qualified certificate.

4.5.8 Qualified Electronic Signature (QES)

A qualified electronic signature is an advanced electronic signature computed over the DTBSR by means of the signer's Signature Creation Data held in the signer's SSCD, which is associated with the signer's relevant Qualified Certificate.

4.5.9 Signed Data Object

This is the output of the SCA produced by the SDOC component and formatted according to the SDO Type. It will contain the digital signature, and may additionally contain the following:

- The SD or SDR;
- The DTBSF;
- Additional supportive unsigned attributes and information such as timestamps or validation data. TS 101 733 [2] and TS 101 903 [1] provide details on unsigned attributes, that are called unsigned properties in TS 101 903.

4.5.10 Signer's Authentication Data (not shown)

This is information supplied by the signer to the SCDev (possibly via the SCA) to authenticate the signer prior to commencement of the signing processes.

4.5.11 Validation data

Some forms of electronic signature incorporate additional data needed for validation. This additional data is called validation data and includes:

- Public Key Certificates (PKCs) and Attributes Certificates (ACs);
- revocation status information for each PKC and AC (Certificate Revocation Lists (CRLs) or certificate status information (OCSP));
- trusted time-stamps applied to the digital signature or a time-mark that shall be available in an audit log;

The validation data may be collected by the signer and/or the verifier.

5 Signature Validation

5.1 Introduction

There are a variety of ways to implement the signature validation procedures, such as

- running as (part of) an application software on a device like a PC with a graphical user interface
- as a web service
- a web application
- a command-line tool
- an integrated library or a middleware for other applications

To cope with this manifold implementation options, this specification uses a simple conceptual model by dividing software with signature validation functions into two parts:

- A signature validation application (SVA) and
- a driving application (DA).

A signature validation application (SVA) receives signed data and other input from the driving application (DA), validates the electronic signature against a set of validation constraints and outputs a validation report. This report consists of a status indication accompanied by additional data items, providing the details of the technical validation of each of the applicable constraints. The status indication can have one of three values:

- VALID: The signature is considered technically valid;
- INVALID: The signature is technically invalid;
- INDETERMINATE: The available information is insufficient to ascertain the signature to be VALID or INVALID.

Clause 5.1.3 details the meaning of the values and requirements on the corresponding validation report. The report **may** include additional information (e.g. explanations and other information to be displayed) that has been found relevant by the SVA and may be relevant for the driving application (DA) in interpreting the results. The output of the SVA is meant to be processed by the DA (e.g. to be displayed to the verifier). Annex E will specify a structure for a signature validation report in a later version of this draft.

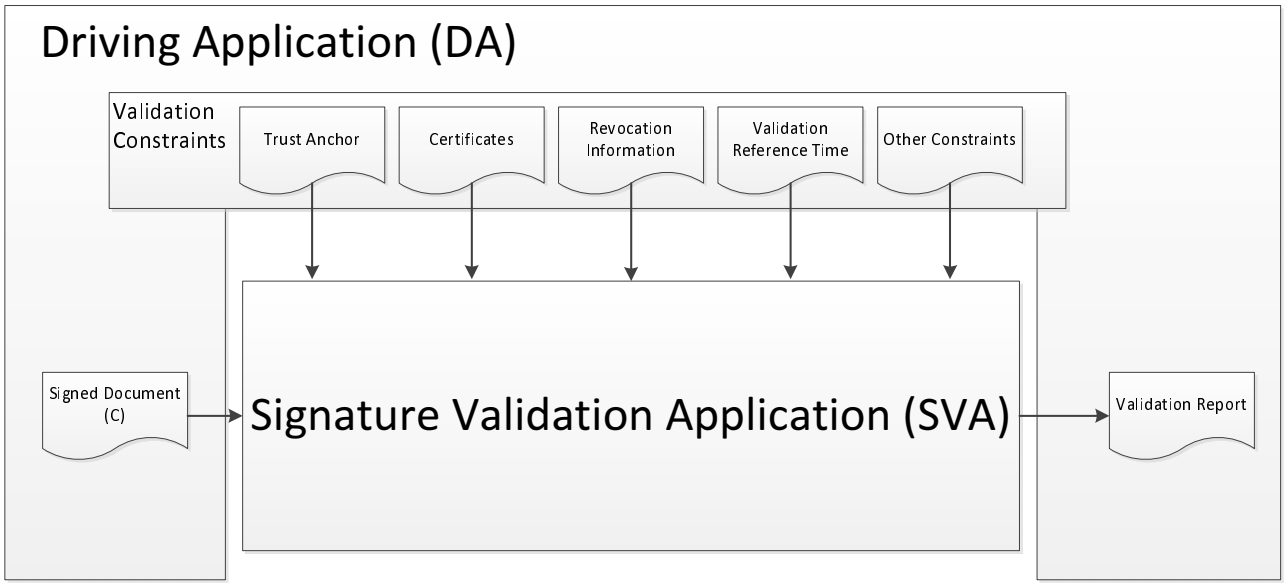


Figure 21: Conceptual Model of Signature Validation

The set of validation constraints used for validation may force the SVA to ignore any condition that otherwise would, according to the present document, require an INVALID or INDETERMINATE result. E.g. if validation constraints force the SVA to ignore revocation status of intermediate certificates, the SVA will return VALID, even if it should return INDETERMINATE. Such overruling by the policy is in theory possible for all decisions made by the present document and cannot be mentioned in all places they may appear. The SVA **shall** report such decisions in the validation report.

Checking that the signature to validate is conformant to the applicable format (e.g. CMS/CAdES, XML-DSig/XAdES, etc.) shall be done by the SVA prior to any subsequent processing. In case the signature is not conformant to the required format, the SVA shall fail with *INVALID/FORMAT_FAILURE* together with details about the format error(s).

> Note: Format checking is out of the scope of the present document. These checks include checking that the syntax of the signature is conformant to the appropriate specification but also any additional checking mandated by that specification for specific signature attributes (e.g. checking that what is time-stamped by a time-stamp token in the signature is really what shall be time-stamped according to the appropriate specification).

The present document does not stipulate any required behaviour by the DA, especially no processing requirements for any of the returned information, since this is application specific and out of the scope of the present document. It is however recommended that:

- If SVA returns VALID for a certain signature, DA should consider the signature as a valid signature according to the validation constraints. This does not necessarily mean that the signature is useful for a particular purpose.
- If SVA returns INVALID or INDETERMINATE, the DA should not consider the signature as a valid signature. In case of INDETERMINATE, the DA may retry verification based on additional information or at a later point of time.

The present document presents the validation process in the form of algorithms to be implemented by a conforming signature validation application. Conforming implementations however are not required to implement these algorithms but shall provide behaviour that is functionally equivalent, i.e. they produce semantically equivalent results given the same set of input information.

The validation constraints against which the signature has to be validated can originate from different sources:

- The signature content itself, either directly (included in the signature or signed attributes) or indirectly, i.e. by reference to an external document, provided either in a human readable and/or machine processable form.
- A local source from the verifier (e.g. configuration file, (machine processable) signature validation policy).

Any format-specific processing is specified in 1.

5.1.1 Types of Validation

Validation of signatures is different, depending on the time of the validation and the form of the signature to validate. We distinguish the following basic validation types:

- Initial Validation: This validation is done on one of the base forms of the signature (BES/EPES) immediately or shortly after creation of the signature. It can be done by the signer or a verifier. Certificate and revocation information collected during that validation may be used to create an extended signature form. Signature and other timestamps may only be applied after successful initial validation.

- Subsequent Validation: This validation type uses references to certificates and revocation information or certificate and revocation information stored within the signature for validation as well as time-stamps protecting signature elements. It may also collect further certificates and revocation information if applicable.

5.1.2 The concept of Proof Of Existence (POE)

A proof of existence is evidence that proves that an object (a certificate, a CRL, signature value, hash value, etc.) existed at a specific date/time, which may be a date/time in the past. The possession of a certain object at current time is a proof of its existence at the current time. A suitable way of providing proof of existence of an object at a time in the past is to generate a time-stamp on that object. Other services can provide proofs of existence by various means (electronic notaries, archival services, etc.).

This concept is used extensively in the clauses below.

5.1.3 Status indication of the signature validation process and Signature Validation Report

A SVA shall provide a comprehensive report of the validation done, allowing the DA to inspect details of the decisions made during validation and investigate the detailed causes for the status indication provided by the SVA. It is clearly out of scope of this standard to specify formats for validation reports and the way the report is provided to the DA. Typically the report is expected to be provided in a structured form. This clause specifies minimum requirements for the content of such a report. The DA must be able to present the report in a way meaningful to the verifier.

In all cases, the signature validation process shall output

- a status indication of the results of the signature validation process. Table 1 lists the possible values of the main status indication and their semantics;

- an indication of the policy or set of constraints against which the signature has been validated;

- the date and time for which the validation status was determined;

- additional validation report data as specified in Table 2 and Table 3.

and **may** output additional data items extracted from the signature.

NOTE 1: The date and time returned will be the current time for basic signature validation; it can be a point in time in the past when validating AdES-T and LTV-Forms.

For the certificate chain validation algorithm, the following assumptions are made:

1. If an intermediate certificate in a chain is revoked, **and if no "better" chain can be found**, a conformant SVA shall return INDETERMINATE, since another chain may exist (that the SVA cannot build due to missing certificates).

5) If a valid chain has been found (certificate path validation procedures defined in [4], clause 6 were successful and none of the intermediate certificates has been revoked) and the signer's certificate is revoked, the chain validation algorithm shall return INDETERMINATE/REVOKED_NO_POE.

NOTE 1: This does not mean that the overall signature validation result will be INVALID. Long term validation may still find the signature to be valid at the time of signing.

Indications returned by SVAs shall conform to the following rules:

- **When the result is due to be VALID or INVALID:**

  a) Any execution of a conformant SVA with the same inputs will return VALID or INVALID, respectively.

  b) Any execution of a conformant SVA with the same inputs + additional validation data (e.g. more certificates) will return the same result as it has returned in *a)* (i.e. VALID or INVALID).

- **When the result is due to be INDETERMINATE:**

  a) Any execution of a conformant SVA with the same inputs will return INDETERMINATE.

  b) Any execution of a conformant SVA with the same inputs + additional validation data will return VALID, INVALID or INDETERMINATE.

NOTE 2: The date/time at which the conformant SVA is executed is an implicit input to the validation process. Subsequent executions of the SVA may give different results in case additional data becomes available (e.g. new certificate status information).

NOTE 3: The term "same inputs" includes the validation constraints to be used. Different validation constraints will in general result in different validation results.

NOTE 4: The status indicators VALID, INVALID and INDETERMINATE are also used in the building blocks specified in the following clauses. For the building-blocks, these statuses only represent the result of the operation performed in the block and not necessarily the result of the overall signature validation. Any sub-indicators used in the building blocks have the semantics of the sub-indicators in Table 3.

Table 2: Status indications of the signature validation process

Status indication	Semantics	Associated Validation report data
VALID	The signature is technically valid based on the following considerations: • The signature is cryptographically valid, and • Any constraints applicable to the signer's identity certification have been positively validated (i.e. the signer's certificate consequently has been found trustworthy), and • The signature has been positively validated against the validation constraints and hence is considered conformant to these constraints.	The validation process shall output the following: • For each of the validation constraints, the result of the validation. • The validated certificate chain, including the signer's certificate, used in the validation process.
INVALID	The signature is invalid based on the failure of at least one of the above considerations.	The validation process shall output additional information to explain the *INVALID* indication for each of the validation constraints that have been taken into account and for which a negative result occurred.
INDETERMINATE	The available information is insufficient to ascertain the signature to be *VALID* or *INVALID*.	The validation process shall output additional information to explain the *INDETERMINATE* indication and to help the Verifier to identify what data is missing to complete the validation process. In particular it shall provide validation result indications for at least those validation constraints that have been taken into account and for which an indeterminate result occurred.

Table 2 gives a recommended structure for the validation report data associated to the *INVALID* and *INDETERMINATE* indications status resulting from the validation of an electronic signature by listing the main sub codes to be returned by the validation process.

Table 3: Validation Report Structure

Main indication	Sub indication	Semantics	Associated Validation report data
INVALID	*REVOKED*	The signature is considered invalid because: • The signer's certificate has been found to be revoked and • The Signature Validation Algorithm can ascertain that the signing time lies after the revocation time.	The validation process shall provide the following: • The certificate chain used in the validation process. • The time and the reason of revocation of the signer's certificate.
	HASH_FAILURE	The signature is considered invalid because at least one hash of a signed data object(s) that has been included in the signing process does not match the corresponding hash value in the signature.	The validation process shall provide: • An identifier (s) (e.g. an URI) uniquely identifying the signed data object that caused the failure.
	SIG_CRYPTO_FAILURE	The signature is considered invalid because the signature value in the signature could not be verified using the signer's public key in the signer's certificate.	The validation process shall output: • The signer certificate used in the validation process.
	SIG_CONSTRAINTS_ FAILURE	The signature is considered invalid because one or more properties of the signature do not match the validation constraints.	The validation process shall provide: • The set of constraints that have not been met by the signature.
	CHAIN_CONSTRAINTS_ FAILURE	The signature is considered invalid because the certificate chain used in the validation process does not match the validation constraints related to the certificate.	The validation process shall output: • The certificate chain used in the validation process. • The set of constraints that have not been met by the chain.
	CRYPTO_CONSTRAINTS_ FAILURE	The signature is considered invalid because at least one of the algorithms that have been used in a material (e.g. the signature value, a certificate...) involved in validating the signature or the size of the keys used with such an algorithm is no longer considered reliable and the Signature Validation Algorithm can ascertain that this material was produced after the time up to which this algorithm was considered secure.	The process shall output: • A list of algorithms, together with the size of the key, if applicable, that have been used in validation of the signature but no longer are considered reliable together with a time up to which each of the listed algorithms were considered secure. • The list of material where each of the listed algorithms were used.
	EXPIRED	The signature is considered invalid because the Signature Validation Algorithm can ascertain that the signing time lies after the expiration date (notAfter) of the signer's certificate.	The process shall output: • The validated certificate chain.
	NOT_YET_VALID	The signature is considered invalid because the Signature Validation Algorithm can ascertain that the signing time lies before the issuance date (notBefore) of the signer's certificate.	

Main indication	Sub indication	Semantics	Associated Validation report data
	FORMAT_FAILURE	The signature has been found not conformant to one of the base standards ([1], [2] and [12] to [15]).	
	POLICY_PROCESSING_ ERROR	A given formal policy file could not be processed for any reason (e.g. not accessible, not parsable, etc.)	The validation process shall provide additional information on the problem.
	UNKNOWN_COMMITMENT _TYPE	The signature was created using a policy and commitment type that is unknown to the SVA.	The validation process shall provide additional information on the problem.
	TIMESTAMP_ORDER_ FAILURE	Some constraints on the order of signature time-stamps and/or signed data object(s) time-stamps are not respected.	The validation process shall output the list of time-stamps that do no respect the ordering constraints.
	GENERIC	Any other reason	The validation process shall output: • The certificate chain used in the validation process. • Additional information why the signature has been declared invalid.
INDETERMINATE	*NO_SIGNER_CERTIFICATE _FOUND*	The signer's certificate cannot be identified.	
	NO_CERTIFICATE_CHAIN_ FOUND	No certificate chain has been found for the identified signer's certificate.	
	REVOKED_NO_POE	The signer's certificate has been found to be revoked at the validation date/time. However, the Signature Validation Algorithm cannot ascertain that the signing time lies before or after the revocation time.	The validation process shall provide the following: • The certificate chain used in the validation process. • The time and the reason of revocation of the signer's certificate.
	REVOKED_CA_NO_POE	At least one certificate chain was found but an intermediate CA certificate has been found to be revoked.	The validation process shall provide the following: • The certificate chain which includes the revoked CA certificate. • The time and the reason of revocation of the certificate.
	OUT_OF_BOUNDS_NO_ POE	The signer's certificate is expired or not yet valid at the validation date/time and the Signature Validation Algorithm cannot ascertain that the signing time lies within the validity interval of the signer's certificate.	
	CRYPTO_CONSTRAINTS_ FAILURE_NO_POE	At least one of the algorithms that have been used in a material (e.g. the signature value, a certificate...) involved in validating the signature or the size of the keys used with such an algorithm is no longer considered reliable at the validation date/time. However, the Signature Validation Algorithm cannot ascertain that the concerned material has been produced before or after the algorithm or the size of the keys have been considered not reliable.	The process shall output: • A list of algorithms, together with the size of the key, if applicable, that have been used in validation of the signature but no longer are considered reliable together with a time up to which each of the listed algorithms were considered secure. The list of material where each of the listed algorithms were used.

Main indication	Sub indication	Semantics	Associated Validation report data
	NO_POE	A proof of existence is missing to ascertain that a signed object has been produced before some compromising event (e.g. broken algorithm).	The validation process should provide additional information on the problem.
	TRY_LATER	Not all constraints can be fulfilled using available information. However, it may be possible to do so using additional revocation information that will be available at a later point of time.	The validation process shall output the point of time, where the necessary revocation information is expected to become available.
	NO_POLICY	The policy to use for validation could not be identified.	
	SIGNED_DATA_NOT_ FOUND	Cannot obtain signed data.	The process should output when available: • The identifier (s) (e.g. an URI) of the signed data that caused the failure.
	GENERIC	Any other reason.	The validation process shall output: Additional information why the validation status has been declared Indeterminate.
	CHAIN_CONSTRAINTS_ FAILURE	The set of certificates available for chain validation produced an error when validating chain constraints.	Additional information why the validation status has been declared Indeterminate
	CERTIFICATE_CHAIN_ GENERAL_FAILURE	The set of certificates available for chain validation produced an error for an unspecified (in this document) reason.	Additional information regarding the reason.

5.1.4 Validation Constraints

The validation process is controlled by a set of validation constraints in use. These constraints may be defined:

- using formal policy specifications, e.g. in one of the standard policy formats [i.2], [i.3]; or
- defined explicitly in system specific control data: e.g. in conventional configuration-files like property or in-files or stored in a registry or database; or
- implicitly by the implementation itself.

Additionally constraints may be provided by the DA to the SVA via parameters implied by the application or the user. This clause defines types of constraints influencing the validation process and the validation result, irrespective of where these constraints have been defined.

Some of the constraints are related to elements of the signature validation process that are widely implemented in applications and already have been standardized elsewhere, e.g. in X.509 or PKIX. Such constraints have been collected in Annex A. Details on how to check that the signature matches such constraints will not be given in the present document. Such standardised constraints are listed in annex A to give an overview of all constraints that are considered relevant for the purpose of the present document. Use of other constraints is outside the scope of the present document.

The verifier may consider additional constraints that are not mentioned in the present document. It is not foreseeable, which constraints a DA may need to impose on the SVA. It is assumed that an implementation handles all constraints properly. If the algorithm prescribes a certain check and the set of constraints state that such a check is not required (e.g. revocation checking), a conformant implementation can skip over that step and assume the check succeeded. In such cases, the SVA shall return, in its final report to the DA, the list of checks that were disabled due to the policy.

The present document does not always prescribe when constraints are to be checked, since this is implementation dependent. A conformant SVA shall however check all constraints that are prescribed.

5.1.5 X.509 certificate meta-data

X.509 certificate meta-data is additional information that is associated to a given certificate, a CRL or an OCSP and that the DA may make available to the SVA.

This is data that may be required to allow the SVA to correctly validate a signature (e.g. to check constraints which are part of a signature validation policy but is not or not easily available to the SVA). Making such meta-data available to the SVA will therefore result more often in a VALID or INVALID response, where the SVA would need to return INDETERMINATE should that information not be available.

NOTE: While some of this meta-data may be retrieved form a Trust-service Status List (TSL) or a Trusted List, the same type of information may be available to the DA in other forms, but are semantically equivalent. For example: If the validation policy requires a qualified certificate, but this information is not contained in the certificate itself, but the certificate is known to be a qualified one, the DA can make this information available to the SVA as meta-data.

Information needed by the DA to build such certificate meta-data may, e.g. be:

- taken from the certificate content TS 101 862 [5], TS 101 456 [7] and TS 102 042 [8];
- derived from a Trust-service Status List [3] entry, or a full Trust-service Status List; or

5.1.6 taken from local configurationTrust Management

While trust management is essential for signature validation, it is out of scope of the present document to define how trust management has to be handled. The X.509 Certificate Validation (XCV)-process as specified in clause 5.3 builds on the Certification Path Validation, as specified in [4], clause 6.1, which is based on trust anchors. Trust anchors are typically retained in the form of (root) certificates that are considered trustworthy, where all certificates issued under such a hierarchy are trusted. The selection of acceptable trust anchors is part of the Validation Context Initialisation (VCI) process when setting up the X.509 Validation Parameters, and it is the responsibility of the DA to select the trust anchors for a validation process.

NOTE: The decision to accept a Certification Authority as a trust anchor is not to be taken lightly. It is a matter of local policy as well as the application context whether a certificate of a CA is acceptable or not. A CA that is trusted for email-exchange may e.g. not be trusted for verification of signed contracts.

How the DA and the SVA agree on which trust anchors are acceptable is implementation dependent and out of scope for the present document. Trust anchors are typically made available as:

- trust points specified in signature validation policies;
- sets of trusted CAs, e.g. represented by their root certificates stored in the environment (like Microsoft's® certificate store); or
- trust service status Lists as specified in [3].

5.1.7 The concept of revocation freshness

To check the revocation status of a certificate at the current time, it is necessary to obtain recent revocation status information about that certificate. However, obtaining revocation status information issued at the current time is (in practice) impossible even with schemes providing real time revocation information (e.g. OCSP). In practice, we use revocation status information issued shortly before the current time and we make the approximation that the information it contains is still reliable at the current time. The freshness of the revocation status information is the maximum accepted difference between the issuance date of the revocation status information and the current time. The nextUpdate field, when present, indicates a date at which a newer CRL should be available; the difference between that value and the thisUpdate field is thus a freshness that should always be fulfillable, and can be used as an upper bound on the freshness that a relying party may require for a given CRL. In general, revocation status information is said "fresh" if its issuance date is after the current time minus the considered freshness.

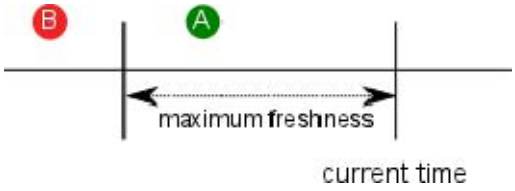


Figure 22: Freshness

Figure 1 shows two objects, A and B, created at the time shown. Object A is considered "fresh", while object B is not, having been created at a time outside the "window of freshness".

The same notion can be extended into the past. When revocation status information is used to ascertain the revocation status of a certificate at a particular date in the past, the revocation status information is said to be "fresh" if it has been issued after the validation date (in the past) minus the considered freshness. See Figure 2 as an illustration for the concept.

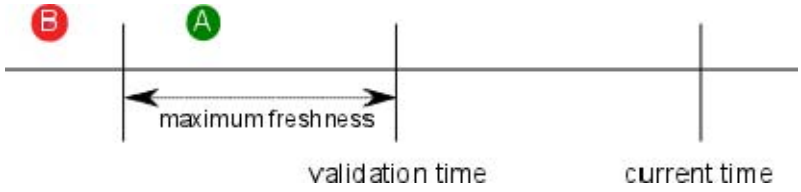


Figure 23: Freshness in the past

5.2 Basic Building Blocks

This clause presents basic building blocks that are useable in the signature validation process. Later clauses will use these blocks to construct validation algorithms for specific scenarios. Figure 3 shows, in a simplified way, how these building are related to achieve signature validation.

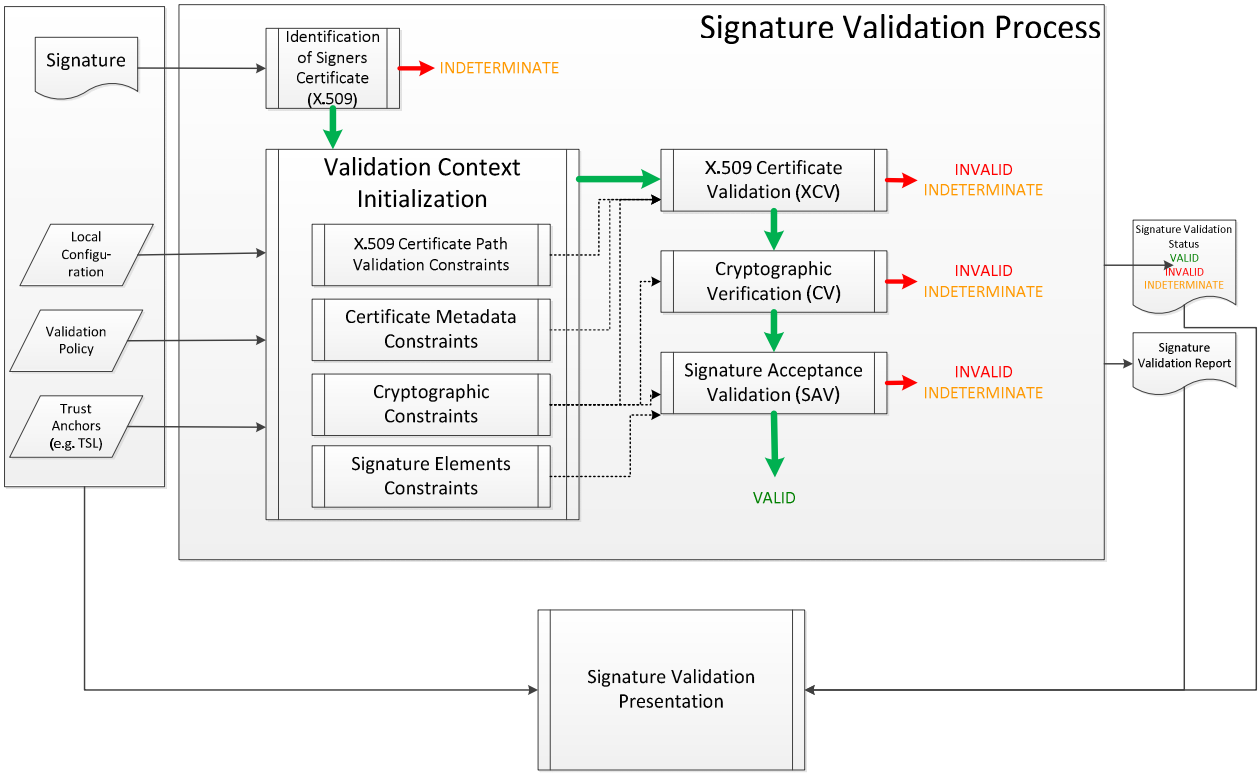


Figure 23: Signature Validation

5.2.1 Identification of the Signer's Certificate (ISC)

5.2.1.1 Description

This process consists in identifying the signer's certificate that will be used to validate the signature.

5.2.1.2 Inputs

Table 4: Inputs to the ISC process

Input	Requirement
Signature	Mandatory
Signer's Certificate	Optional

5.2.1.3 Outputs

- In case of success, i.e. the signer's certificate can be identified, the output shall be the signer's certificate.
- In case of failure, i.e. the signer's certificate cannot be identified, the output shall be the indication *INDETERMINATE* and the sub indication *NO_SIGNER_CERTIFICATE_FOUND.*

NOTE: If the signature creation process has been compliant with this document or the CD 2011/130/EU, this process will never return *INDETERMINATE*, since the signer's certificate is present in the signature.

5.2.1.4 Processing

The common way to unambiguously identify the signer's certificate is by using a property/attribute of the signature containing a reference to it (see clause 4.5.2.1). The certificate can either be found in the signature or it can be obtained using external sources. The signer's certificate may also be provided by the DA. If the certificate cannot be retrieved, the indication *INDETERMINATE* will be the result.

The signing certificate shall be checked against all references present in the signature attributes, since one of these references shall be a reference to the signing certificate [4.5.2.1]. The following steps shall be performed:

1. Take the first reference and check that the digest of the certificate referenced matches the result of digesting the signing certificate with the algorithm indicated. If they do not match, take the next element and repeat this step until a matching element has been found or elements have been checked. If they do match, continue with step 2. If the last element is reached without finding any match, the validation of this property shall be taken as failed and *INVALID/FORMAT_FAILURE* is returned.
2. If the issuer and the serial number indicated in that element and the signing certificate do not match, the validation of this property shall be taken as failed and *INDETERMINATE* is returned.
3. Otherwise, return the signer’s certificate

NOTE: If the signature format used contains a way to directly identify the reference to the signers’ certificate in the attribute, only that certificate needs to be checked in step 1.

5.2.2 Validation Context Initialization (VCI)

5.2.2.1 Description

This process consists in initializing the validation constraints (chain constraints, cryptographic constraints, signature elements constraints) and parameters (X.509 validation parameters, certificate meta-data) that will be used to validate the signature. The constraints and parameters may be initialized from any of the sources listed in clauses 5.1.4, 5.1.5 and 5.1.6.

5.2.2.2 Inputs

Table 5: Inputs to the VCI process

Input	Requirement
Signature	Mandatory
Signature Validation Policies	Optional
Trusted-status Service Lists	Optional
Local configuration	Optional

5.2.2.3 Outputs

In case of failure, the process outputs *INDETERMINATE* or *INVALID* with an indication explaining the reason(s) of failure.

In case of success, the process outputs the following:

Table 6: Output of the VCI process

Output
X.509 Validation Constraints
Certificate Meta-data Constraints
Chain Constraints
Cryptographic Constraints
Signature Elements Constraints

5.2.2.4 Processing

If the validation constraints have been initialized using an allowed set of signature validation policies [i.2], [i.3] and if the signature has been created under one of these policies and also contains a commitment type indication property/attribute, the specific commitment defined in the policy shall be selected using this attribute. The clauses below describe the processing of these properties/attributes. The processing of additional sources for initialization (e.g. local configuration) is out of the scope of the present document.

This implies that a signature policy referenced in a signature is expected to be known to the verifier and listed in the set of acceptable policies. If the policy is unknown to the verifier, accepting a commitment type is not possible and may even be dangerous. In this case, the SVA shall return *INVALID/UNKNOWN_COMMITMENT_TYPE*.

If the SVA cannot access a file representing the policy, the policy is not able to parse the policy file or the SVA cannot process the policy for any other reason, it shall return INVALID/POLICY_PROCESSING_ERROR with an appropriate indication. If the SVA cannot identify the policy to use, it shall return INDETERMINATE/ NO_POLICY.

5.2.2.4.1 Processing commitment type indication

If this signed property is present, it allows identifying the commitment type and thus affects all rules for validation, which depend on the commitment type that shall be used in the validation context initialization.

5.2.2.4.2 Processing Signature Policy Identifier

If this signed property/attribute is present and it is not implied, the SVA shall perform the following checks. If any of these checks fail, then the SVA shall assume that a failure has occurred during the verification and return *INVALID/ POLICY_PROCESSING_ERROR* with an indication that the validation failed to an invalid signature policy identifier property/attribute.

1. Access the electronic document identified by the contents of the property/attribute and containing the details of the policy; if it is not available, cannot be parsed or processed for any other reason: terminate with *INDETERMINATE/SIGNATURE_POLICY_NOT_AVAILABLE*.

2. Calculate the digest of the resulting document using the algorithm specified in the property/attribute.

3. Check that the digest obtained in the previous step is equal to the digest value indicated in the property/attribute.

4. Should the property/attribute have qualifiers, manage them according to the rules that are stated by the policy applying within the specific scenario.

5. If the checks described before end successfully, the process extracts the validation constraints from the rules encoded in the validation policy. If an explicit commitment is identified, select the rules corresponding to this commitment in the signature. If the commitment is not recognized, the Verifier may select the rules dependent on other sources (e.g. the data being signed). The way used by the signature policy for presenting the rules and their description are out of the scope of the present document. TR 102 038 [i.3] specifies a "XML format for signature policies" that may be automatically processed.

If the signature policy is implied, and stated so by the signature rules, the SVA shall perform the checks mandated by the implicit signature policy that shall be provided by the verifier by one of the methods described in clause 4.2.

NOTE: An implicit policy can in the most general case either be established according to the minimum requirements by law or if being more constrained only be discovered in well-known or pre-agreed (driving) application contexts.

5.2.3 X.509 Certificate Validation (XCV)

5.2.3.1 Description

The objective of this process is to validate the signer's certificate.

5.2.3.2 Inputs

Table 7: Inputs to the XCV process

Input	Requirement
Signer's certificate	Mandatory
X.509 Validation Constraints	Mandatory
Certificate Meta-data Constraints	Optional
Chain Constraints	Optional
Cryptographic Constraints	Optional
Other Certificates	Optional

Note: Any certificates stored in the signature have to be passed in “Other Certificates”.

5.2.3.3 Outputs

The process outputs one of the following indications together with the associated validation report data.

Table 8: Output of the XCV process

Indication	
VALID	
INDETERMINATE	*NO_CERTIFICATE_CHAIN_FOUND*
	OUT_OF_BOUNDS_NO_POE
	REVOKED_NO_POE
	CRYPTO_CONSTRAINTS_FAILURE_NO_POE
	INDETERMINED/TRY_LATER + Next_update
	REVOKED_CA_NO_POE
INVALID	*CHAIN_CONSTRAINTS_FAILURE*

5.2.3.4 Processing

This process consists of the following steps:

1. Check that the current time is in the validity range of the signer's certificate. If this constraint is not satisfied, abort the processing with the indication *INDETERMINATE* and the sub indication *OUT_OF_BOUNDS_NO_POE*.

2. Build a new prospective certificate chain that has not yet been evaluated. If the OtherCertificates parameter is present, only certificates contained in that set of certificates may be used to build the chain. The chain shall satisfy the conditions of a prospective certificate chain as stated in [4], clause 6.1, using one of the trust anchors provided in the inputs:

   a) If no new chain can be built, abort the processing with the current status and the last chain built or, if no chain was built, with *INDETERMINATE/NO_CERTIFICATE_CHAIN_FOUND*.

   b) Otherwise, add this chain to the set of prospected chains and go to step 3.

3. Run the Certification Path Validation [4], clause 6.1, with the following inputs: the prospective chain built in the previous step, the trust anchor used in the previous step, the X.509 parameters provided in the inputs and the current date/time. The validation shall include revocation checking for each certificate in the chain:

   a) If the certificate path validation returns a success indication and the revocation information used is considered fresh, go to the next step.

   b) If the certificate path validation returns a success indication and the revocation information used is not considered fresh, abort the process with the indication *INDETERMINATE*, the sub indication *TRY_LATER* and the content of the NEXT_UPDATE-field of the CRL used as the suggestion for when to try the validation again.

   c) If the certificate path validation returns a failure indication because the signer's certificate has been determined to be revoked, abort the process with the indication *INDETERMINATE*, the sub indication *REVOKED_NO_POE*, the validated chain, the revocation date and the reason for revocation.

   d) If the certificate path validation returns a failure indication because the signer's certificate has been determined to be on hold, abort the process with the indication *INDETERMINATE*, the sub indication *TRY_LATER*, the suspension time and, if available, the content of the NEXT_UPDATE-field of the CRL used as the suggestion for when to try the validation again.

   e) If the certificate path validation returns a failure indication because an intermediate CA has been determined to be revoked, set the current status to *INDETERMINATE/REVOKED_CA_NO_POE* and go to step 2.

   f) If the certificate path validation returns a failure indication with any other reason, set the current status to *INDETERMINATE/CERTIFICATE_CHAIN_GENERAL_FAILURE* and go to step 2.

4. Apply the Chain Constraints to the chain. Certificate meta-data constraints shall be taken into account when checking these constraints against the chain. If the chain does not match these constraints, set the current status to INVALID/CHAIN_CONSTRAINTS_FAILURE and go to step 2.

5. Apply the cryptographic constraints to the chain. If the chain does not match these constraints, set the current status to INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE and go to step 2.

6. Return the chain with the indication VALID.

NOTE 1: Chain construction (step 2) and validation (step 3) may use validation data (certificates, CRLs, etc.) extracted from the signature or obtained from other sources (e.g. LDAP servers). The management of the sources for the retrieval of validation data is out of the scope of the present document.

NOTE 2: For more information and rational about certificate chain construction, refer to [i.1].

5.2.4 Cryptographic Verification (CV)

5.2.4.1 Description

This process consists in verifying the integrity of the signed data by performing the cryptographic verifications.

5.2.4.2 Inputs

Table 9: Inputs to the CV process

Input	Requirement
Signature	Mandatory
Signer Certificate	Mandatory
Validated certificate chain	Optional
Signed data object(s)	Optional

NOTE: In most cases, the cryptographic verification requires only the signer's certificate and not the entire validated chain. However, for some algorithms the full chain may be required (e.g. the case of DSS/DSA public keys which inherit their parameters from the issuer certificate).

5.2.4.3 Outputs

The process outputs one of the following indications together with the associated validation report data:

Table 10: Outputs of the CV process

Indication		Description	Additional data items
VALID		The signature passed the cryptographic verification.	
INVALID	*HASH_FAILURE*	The hash of at least one of the signed data items does not match the corresponding hash value in the signature.	The process **should** output: • The identifier (s) (e.g. an URI) of the signed data that caused the failure.
	SIG_CRYPTO_FAILURE	The cryptographic verification of the signature value failed.	
INDETERMINATE	*SIGNED_DATA_NOT_FOUND*	Cannot obtain signed data.	The process **should** output: • The identifier (s) (e.g. an URI) of the signed data that caused the failure.

5.2.4.4 Processing

The first and second steps as well as the Data To Be Signed depend on the signature type. The technical details on how to do this correctly are out of scope for the present document. See [10], [16], [12], [13], [14] and [15] for details:

1. Obtain the signed data objects(s) if not provided in the inputs (e.g. by dereferencing an URI present in the signature). If the signed data object (s) cannot be obtained, abort with the indication *INDETERMINATE/SIGNED_DATA_NOT_FOUND*.

2. Check the integrity of the signed data objects. In case of failure, abort the signature validation process with *INVALID/HASH_FAILURE*.

3. Verify the cryptographic signature using the public key extracted from the signer's certificate in the chain, the signature value and the signature algorithm extracted from the signature. If this cryptographic verification outputs a success indication, terminate with *VALID*. Otherwise, terminate with *INVALID/SIG_CRYPTO_FAILURE*.

5.2.5 Signature Acceptance Validation (SAV)

5.2.5.1 Description

This building block covers any additional verification that shall be performed on the attributes/properties of the signature.

5.2.5.2 Inputs

Table 11: Inputs to the SAV process

Input	Requirement
Signature	Mandatory
Cryptographic verification output	Optional
Cryptographic Constraints	Optional
Signature Elements Constraints	Optional

5.2.5.3 Outputs

The process outputs one of the following indications:

Table 12: Outputs of the SVA process

Indication		Description	Additional data items
VALID		The signature is conformant with the validation constraints.	
INVALID	*SIG_CONSTRAINTS_FAILURE*	The signature is not conformant with the validation constraints.	The process shall output: • The set of constraints that are not verified by the signature.
INDETERMINATE	*CRYPTO_CONSTRAINTS_FAILURE_NO_POE*	At least one of the algorithms used in validation of the signature together with the size of the key, if applicable, used with that algorithm is no longer considered reliable.	The process shall output: • A list of algorithms, together with the size of the key, if applicable, that have been used in validation of the signature but no longer are considered reliable together with a time up to which each of the listed algorithms were considered secure.

5.2.5.4 Processing

This process consists in checking the Signature and Cryptographic Constraints against the signature. The general principle is as follows: perform the following for each constraint:

- If the constraint necessitates processing a property/attribute in the signature, perform the processing of the property/attribute as specified from clauses 5.2.5.4.1 to 5.2.5.4.8.

- If at least one of the algorithms that have been used in validation of the signature or the size of the keys used with such an algorithm is no longer considered reliable, return *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE* together with the list of algorithms and key sizes, if applicable, that are concerned and the time for each of the algorithms up to which the respective algorithm was considered secure.

NOTE 1: We do that, since the algorithm or key size used may at the time of signing the signed object have been perfectly secure and only expired years later. Long term validation may then still allow validation of the signed object if e.g. time-stamps using different, still secure, algorithms or key sizes have been applied in time. E.g. an RSA-key of 2 400 bits is currently assumed to be secure for ~20 years. If a signature created with such a key has to be verified using this algorithm in 25 years from now, it can be secured by e.g. creating a time-stamp using an RSA-key of ~5 300 bits [i.5]. The algorithms of concern are not only the hash- and signature-algorithm for the signature itself, but also for any of the Certificate, CRLs, time-stamps or other material used in the validation process.

- If one or more checks fail, output *INVALID/SIG_CONSTRAINTS_FAILURE* together with the set of constraints that are not satisfied by the signature.

- If all the constraints are satisfied, output *VALID*.

NOTE 2: The SVA **may** ignore processing a property/attribute for which no validation constraint is specified.

5.2.5.4.1 Processing AdES properties/attributes

This clause describes the application of Signature Elements Constraints on the content of the signature including the processing on signed and unsigned properties/attributes.

5.2.5.4.2 Processing signing certificate reference constraint

If the `SigningCertificate` property contains references to other certificates in the path, the verifier shall check each of the certificates in the certification path against these references as specified in 5.2.1.

Should this property contain one or more references to certificates other than those present in the certification path, the verifier shall assume that a failure has occurred during the verification.

Should one or more certificates in the certification path not be referenced by this property, the verifier shall assume that the verification is successful unless the signature policy mandates that references to all the certificates in the certification path "shall" be present.

5.2.5.4.3 Processing claimed signing time

If the signature elements constraints contain constraints regarding this property, the verifying application shall follow its rules for checking this signed property.

Otherwise, the verifying application shall make the value of this property/attribute available to its DA, so that it may decide additional suitable processing, which is out of the scope of the present document.

5.2.5.4.4 Processing signed data object format

If the signature elements constraints contain constraints regarding this property, the verifying application shall follow its rules for checking this signed property.

Otherwise, the verifying application shall make the value of this property/attribute available to the DA, so that it may decide additional suitable processing, which is out of the scope of the present document.

5.2.5.4.5 Processing indication of production place of the signature

If the signature elements constraints contain constraints regarding this property, the verifying application shall follow its rules for checking this signed property.

Otherwise, the verifying application shall make the value of this property/attribute available to its DA, so that it may decide additional suitable processing, which is out of the scope of the present document.

5.2.5.4.6 Processing time-stamps on signed data objects

If the signature elements constraints contain specific constraints for content-time-stamp attributes, the SVA shall check that they are satisfied. To do so, the SVA shall do the following steps for each content-time-stamp attribute:

1. Perform the Validation Process for AdES time-Stamps as defined in clause 5.4 with the time-stamp token of the content-time-stamp attribute.

2. Check the message imprint: check that the hash of the signed data obtained using the algorithm indicated in the time-stamp token matches the message imprint indicated in the token.

3. Apply the constraints for content-time-stamp attributes to the results returned in the previous steps. If any check fails, return *INVALID/SIG_CONSTRAINTS_FAILURE* with an explanation of the unverified constraint.

5.2.5.4.7 Processing Countersignatures

If the signature elements constraints define specific constraints for countersignature attributes, the SVA shall check that they are satisfied. To do so, the SVA shall do the following steps for each countersignature attribute:

1. Perform the validation process for AdES-BES/EPES using the countersignature in the property/attribute and the signature value octet string of the signature as the signed data object.

2. Apply the constraints for countersignature attributes to the result returned in the previous step. If any check fails, return *INVALID/SIG_CONSTRAINTS_FAILURE* with an explanation of the unverified constraint.

If the signature elements constraints do not contain any constraint on countersignatures, the SVA **may** still verify the countersignature and provide the results in the validation report. However, it shall **not** consider the signature validation to having failed if the countersignature could not be verified.

5.2.5.4.8 Processing signer attributes/roles

If the signature elements constraints define specific constraints for certified attributes/roles, the SVA shall perform the following checks:

1. The SVA shall verify the validity of the attribute certificate(s) present in this property/attribute following the rules established in [6].

2. The SVA shall check that the attribute certificate(s) actually match the rules specified in the input constraints.

If the signature rules do not specify rules for certified attributes/roles, the SVA shall make the value of this property/attribute available to its DA so that it may decide additional suitable processing, which is out of the scope of the present document.

5.2.6 Signature Validation Presentation Component (SVP)

The *Signature Validation Presentation Component* is an optional element in the signature validation process that can be used by a validator to check the results of a validation process. The SVP should support

- Presenting the data (SD) that has been covered by the signature. This can be done by using a SD Presentation Components (see clause 4.3.2);

- Presenting information identifying the signer; Present the date and time for which the validation status was determined

- Presenting any signature attributes that have been included in the signature and make clear which attributes were signed and which were unsigned:

- Making clear which Signature Validation Policy has been used for validation:

- Presenting the overall status of the signature validation (VALID, INVALID, INDETERMINATE);

- In case of INVALID: Present the the reason for the signature being invalid;

- In case of INDETERMINATE: Highlight the parts of the validation report that indicates steps to be taken to potentially get to a determinate result;
- Presenting the validation report.

5.3 Basic Validation Process

5.3.1 Description

This clause describes a validation process for basic short-term signature validation that is appropriate for validating basic signatures (e.g. time-stamps, CRLs, etc.) as well as AdES-BES and AdES-EPES electronic signatures. The process is built on the building blocks described in the previous clause.

5.3.2 Inputs

Table 13: Inputs to BES/EPES validation

Input	Requirement
Signature	Mandatory
Signed data object (s)	Optional
Signer's Certificate	Optional
Trusted-status Service Lists	Optional
Signature Validation Policies	Optional
Local configuration	Optional

5.3.3 Outputs

The main output of the signature validation is a status indicating the validity of the signature. This status may be accompanied by additional information (see clause 4).

5.3.4 Processing

NOTE 1: Since processing is largely implementation dependent, the steps listed in this clause are not necessarily to be processed exactly in the order given. Any ordering that produces the same results can be used, even parallel processing is possible.

The following steps shall be performed:

1. Identify the signer's certificate: Perform the Signer's Certificate Identification process (see clause 5.2.1) with the signature and the signer's certificate, if provided as a parameter. If it returns *INDETERMINATE*, terminate with *INDETERMINATE* and associated information, otherwise go to the next step.
2. Initialize the validation constraints and parameters: Perform the Validation Context Initialization process (see clause 5.2.2).
3. Validate the signer's certificate: Perform the X.509 Certificate Validation process (see clause 5.2.3) with the following inputs:
   a) The signature.
   b) The signer's certificate obtained in step 1.
   c) X.509 Validation Parameters, Certificate meta-data, Chain Constraints and Cryptographic Constraints obtained in step 2:
      - If the process returns *VALID*, go to the next step.

- If the process returns *INDETERMINATE/REVOKED_NO_POE*: If the signature contains a content-time-stamp attribute, perform the Validation Process for AdES time-Stamps as defined in clause 5.4. If it returns *VALID* and the generation time of the time-stamp token is after the revocation time, terminate with *INVALID/REVOKED*. In all other cases, terminate with *INDETERMINATE/REVOKED_NO_POE*.

- If the process returns *INDETERMINATE/OUT_OF_BOUNDS_NO_POE*: If the signature contains a content-time-stamp attribute, perform the Validation Process for AdES time-Stamps as defined in clause 5.4. If it returns *VALID* and the generation time of the time-stamp token is after the expiration date of the signer's certificate, terminate with *INVALID/EXPIRED*. In all other cases, terminate with *INDETERMINATE/OUT_OF_BOUNDS_NO_POE*.

- In all other cases, terminate with the returned indication and associated information.

4. Verify the cryptographic signature value: Perform the Cryptographic Verification process with the following inputs:

   a) The signature.

   b) The certificate chain returned in the previous step.

   c) The signed data object(s).

If the process returns *VALID*, go to the next step. Otherwise, terminate with the returned indication and associated information.

5. Apply the validation constraints: Perform the Signature Acceptance Validation process with the following inputs:

   a) The signature.

   b) The Cryptographic Constraints.

   c) The Signature Elements Constraints.

- If the process returns *VALID*, go to the next step.

- If the process returns INDETERMINATE/*CRYPTO_CONSTRAINTS_FAILURE_NO_POE* and the material concerned by this failure is the signature value: If the signature contains a content-time-stamp attribute, perform the Validation Process for AdES time-Stamps as defined in clause 5.4. If it returns *VALID* and the algorithm(s) concerned were no longer considered reliable at the generation time of the time-stamp token, terminate with *INVALID/CRYPTO_CONSTRAINTS_FAILURE*. In all other cases, terminate with *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE*.

NOTE 2: The content time-stamp is a signed attribute and hence proves that the signature value was produced after the generation time of the time-stamp token.

NOTE 3: In case this clause returns *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE*, LTV can be used to validate the signature, if other POE (e.g. from a trusted archive) exist.

- In all other cases, terminate with the returned indication and associated information.

6. If the policy prescribes a grace period (see 4.1.1.1), and the grace period has not passed, return INDETERMINATE/GRACE_PERIOD_NOT_REACHED with a suggestion, when the signature validation should be retried. In addition, the SVA **should** return additional information as suggested in step 7.

7. Data extraction: the SVA shall return the success indication *VALID*. In addition, the SVA **should** return additional information extracted from the signature and/or used by the intermediate steps. In particular, the SVA **should** provide to the DA all information related to signed and unsigned properties/attributes, including those which were not processed during the validation process. What the DA shall do with this information is out of the scope of the present document.

5.4 Validation Process for Time-Stamps

5.4.1 Description

This clause describes a process for the validation of an RFC 3161 [11] time-stamp token.

An RFC 3161 [11] time-stamp token is basically a CAdES-BES signature. Hence, the validation process is built in the validation process of a CAdES-BES signature.

5.4.2 Inputs

Table 14: Inputs to time-stamp validation

Input	Requirement
Time-stamp token	Mandatory
Trusted-status Service Lists	Optional
Signature Validation Policies	Optional
Local configuration	Optional
Time-Stamp Certificate	Optional

5.4.3 Outputs

The main output of the signature validation is a status indicating the validity of the signature. This status may be accompanied by additional information (see clause 4).

5.4.4 Processing

The following steps shall be performed:

1. Token signature validation: perform the validation process for BES signatures (see clause 5.3) with the time-stamp token. In all the steps of this process, take into account that the signature to validate is a time-stamp token (e.g. to select TSA trust-anchors). If this step ends with a success indication, go to the next step. Otherwise, fail with the indication and information retuned by the validation process.

2. Data extraction: in addition to the data items returned in step 1, the process shall return data items extracted from the TSTInfo [11] (the generation time, the message imprint, etc.). These items may be used by the SVA in the process of validating the AdES signature.

5.5 Validation Process for AdES-T

5.5.1 Description

An AdES-T signature is built on BES or EPES signature and incorporates trusted time associated to the signature. The trusted time may be provided by two different means:

- A signature time-stamp unsigned property/attribute added to the electronic signature.
- A time mark of the electronic signature provided by a trusted service provider.

This clause describes a validation process for AdES-T signatures.

5.5.2 Inputs

Table 15: Inputs to AdES-T validation

Input	Requirement
Signature	Mandatory
Signed data object (s)	Optional
Trusted-status Service Lists	Optional
Signature Validation Policies	Optional
Local configuration	Optional
Signer's Certificate	Optional

5.5.3 Outputs

The main output of the signature validation is a status indicating the validity of the signature. This status may be accompanied by additional information (see clause 4).

5.5.4 Processing

The following steps shall be performed:

1. Initialize the set of signature time-stamp tokens from the signature time-stamp properties/attributes present in the signature and initialize the best-signature-time to the current time.

NOTE 1: Best-signature-time is an internal variable for the algorithm denoting the earliest time when it can be proven that a signature has existed.

2. Signature validation: Perform the validation process for BES signatures (see clause 5.3) with all the inputs, including the processing of any signed attributes/properties as specified. If this validation outputs *VALID*, *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE*, *INDETERMINATE/REVOKED_NO_POE or INDETERMINATE/OUT_OF_BOUNDS_NO_POE*, go to the next step. Otherwise, terminate with the returned status and information.

NOTE 2: We continue the process in the case *INDETERMINATE/REVOKED_NO_POE*, because a proof that the signing occurred before the revocation date may help to go from INDETERMINATE to *VALID* (step 5-a).

NOTE 3: We continue the process in the case *INDETERMINATE/OUT_OF_BOUNDS_NO_POE*, because a proof that the signing occurred before the issuance date (notBefore) of the signer's certificate may help to go from *INDETERMINATE* to *INVALID* (step 5-b).

NOTE 4: We continue the process in the case *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE*, because a proof that the signing occurred before the time one of the algorithms used was no longer considered secure may help to go from INDETERMINATE to VALID (step 5-c).

3. Verification of time-marks: the verification of time-marks is out of the scope of the present document. If the SVA accepts a time-mark as trustworthy (based on out-of-band mechanisms) and if the indicated time is before the best-signature-time, set best-signature-time to the indicated time.

4. Signature time-stamp validation: Perform the following steps:

   a) Message imprint verification: For each time-stamp token in the set of signature time-stamp tokens, do the message imprint verification as specified in clauses 5.4.4 . If the verification fails, remove the token from the set.

   b) Time-stamp token validation: For each time-stamp token remaining in the set of signature time-stamp tokens, the SVA shall perform the time-stamp validation process (see clause 5.4):

      - If *VALID* is returned and if the returned generation time is before best-signature-time, set best-signature-time to this date and try the next token.

- If no specific constraints mandating the validity of the attribute are specified in the validation constraints, remove the time-stamp token from the set of signature time-stamp tokens and try the next token.
- Otherwise, fail with the returned indication/subcode and associated explanations.

5. Comparing times:

   a) If step 2 returned *INDETERMINATE/REVOKED_NO_POE:* If the returned revocation time is posterior to best-signature-time, perform step 5d. Otherwise, terminate with *INDETERMINATE/REVOKED_NO_POE.* In addition to the data items returned in steps 1 and 2, the SVA **should** notify the DA with the reason of the failure.

   b) If step 2 returned *INDETERMINATE/OUT_OF_BOUNDS_NO_POE:* If best-signature-time is before the issuance date of the signer's certificate, terminate with INVALID/NOT_YET_VALID. Otherwise, terminate with *INDETERMINATE/OUT_OF_BOUNDS_NO_POE.* In addition to the data items returned in steps 1 and 2, the SVA **should** notify the DA with the reason of the failure.

   c) If step 2 returned *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE* and the material concerned by this failure is the signature value or a signed attribute, check, if the algorithm(s) concerned were still considered reliable at best-signature-time, continue with step d. Otherwise, terminate with *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE.*

   d) For each time-stamp token remaining in the set of signature time-stamp tokens, check the coherence in the values of the times indicated in the time-stamp tokens. They shall be posterior to the times indicated in any time-stamp token covered by the time-stamp. The SVA shall apply the rules specified in RFC 3161 [11], clause 2.4.2 regarding the order of time-stamp tokens generated by the same or different TSAs given the `accuracy` and `ordering` fields' values of the `TSTInfo` field, unless stated differently by the signature constraints. If all the checks end successfully, go to the next step. Otherwise return *INVALID/TIMESTAMP_ORDER_FAILURE.*

6. Handling Time-stamp delay: If the validation constraints specify a time-stamp delay, do the following:

   a) If no signing-time property/attribute is present, fail with *INDETERMINATE/ SIG_CONSTRAINTS_FAILURE* and an explanation that the validation failed due to the absence of claimed signing time.

   b) If a signing-time property/attribute is present, check that the claimed time in the attribute plus the time-stamp delay is after the best-signature-time. If the check is successful, go to the next step. Otherwise, fail with *INVALID/SIG_CONSTRAINTS_FAILURE* and an explanation that the validation failed due to the time-stamp delay constraint.

7. Data extraction: the SVA shall return the success indication *VALID.* In addition, the SVA **should** return additional information extracted from the signature and/or used by the intermediate steps. In particular, the SVA should return intermediate results such as the validation results of any signature time-stamp token or time-mark. What the DA does with this information is out of the scope of the present document.

NOTE 5: In the algorithm above, the signature-time-stamp protects the signature against the revocation of the signer's certificate (step 5-a) but not against expiration. The latter case requires validating the signer's certificate in the past (see clause 5.6).

5.6 Validation of LTV forms

This clause describes a validation process for signatures with long-term validation (LTV) information that is appropriate for validating AdES-A as well as any intermediate form (e.g. AdES-C, AdES-XL, etc.). The process described in this clause can also be used to validate basic signatures (e.g. AdES-BES and AdES-EPES).

In particular, this is useful in the case where the SVA shall take as input, in addition to the basic signature to validate, additional evidences derived from previous validation (e.g. a proof of existence derived from the validation of a time-stamp token). Such a validation may be done off-line when all required validation material is available within the signature and local configuration. The process is built on the building block described in clause 5 and the additional building blocks defined in clause 5.6.1.

5.6.1 Additional Building blocks

5.6.1.1 Past certificate validation

5.6.1.1.1 Description

This process validates a certificate at a date/time which may be in the past. This may become necessary in the LTV settings when a compromising event (for instance, the end-entity certificate expires) prevents the traditional certificate validation algorithm (see clause 5.2.3) to asserting the validation status of a certificate (for instance, in case the end-entity certificate is expired at the current time, the traditional validation algorithm will return *INDETERMINATE/OUT_OF_BOUNDS_NO_POE* due to the step 1).

The rationale of the algorithm described below are given in [i.4] and can be summarized in the following: if a certificate chain has been useable to validate a certificate at some date/time in the past, the same chain can be used at the current time to derive the same validity status, provided each certificate in the chain satisfies one of the following:

a) The revocation status of the certificate can be ascertained at the current time (typically if the certificate is not yet expired and appropriate revocation status information is obtained at the current time).

b) The revocation status of the certificate can be ascertained using "old" revocation status information such that the certificate (resp. the revocation status information) is proven to having existed at a date in the past when the issuer of the certificate (resp. the revocation status information) was still considered reliable and under control of its signing key. This particular date/time will be named *control-time*.

NOTE: Control-time is an internal variable that is used within the algorithms and not part of the core results of the validation process.

Assuming that the trust anchor is still accepted as such at current time, the validation process will slide the control-time from the current-time to some date in the past each time it encounters a certificate proven to be revoked. In addition to the certificate chain, the process outputs the last value of control-time - the control-time associated with the target certificate (the certificate to validate) which is a point in time when all certificates in the chain were valid. Any object signed with the target certificate and proven to exist before this control-time can be accepted as *VALID*. This assertion is the basis of the LTV validation processes presented in the next clauses. For more readability, the sliding algorithm is presented in its own building block (control-time sliding process) described in the next clause.

It is important to note that when all the certificates in the chain can be validated at the current time, the control-time never slides and the algorithm boils down to the traditional certificate validation algorithm described in clause 5.2.3.

The process below builds a prospective certificate chain in a very same way as in clause 5.2.3except that the X.509 validation algorithm is performed at a determined date in the past (instead of the current date/time) and without any revocation checking. For each such chain, the sliding algorithm is executed to calculate the control-time.

5.6.1.1.2 Input

Input	Requirement
Signature or time-stamp token	Mandatory
Target certificate	Mandatory
X.509 Validation Parameters	Mandatory
A set of POEs	Mandatory
Certificate meta-data	Optional
Chain Constraints	Optional
Cryptographic Constraints	Optional

5.6.1.1.3 Output

Indication	
VALID	
INDETERMINATE	*CHAIN_CONSTRAINTS_FAILURE*
	NO_CERTIFICATE_CHAIN_FOUND
	NO_POE

5.6.1.1.4 Processing

The following steps shall be performed:

1. Build a new prospective certificate chain that has not yet been evaluated. The chain shall satisfy the conditions of a prospective certificate chain as stated in [4], clause 6.1, using one of the trust anchors provided in the inputs:

   a) If no new chain can be built, abort the processing with the current status and the last chain built or, if no chain was built, with *INDETERMINATE/NO_CERTIFICATE_CHAIN_FOUND.*

   b) Otherwise, go to the next step.

2. Run the Certification Path Validation [4], clause 6.1, with the following inputs: the prospective chain built in the previous step, the trust anchor used in the previous step, the X.509 parameters provided in the inputs and a date from the intersection of the validity intervals of all the certificates in the prospective chain. The validation shall **not** include revocation checking:

   a) If the certificate path validation returns a success indication, go to the next step.

   b) If the certificate path validation returns a failure indication because an intermediate CA has been determined to be revoked, set the current status to *INDETERMINATE/REVOKED_CA_NO_POE* and go to step 1.

   c) If the certificate path validation returns a failure indication with any other reason, set the current status to *INDETERMINATE/CERTIFICATE_CHAIN_GENERAL_FAILURE* and go to step 1

   b) If the certificate path validation returns any other failure indication, go to step 1.

3. Perform the control-time sliding process with the following inputs: the prospective chain, the set of POEs and the cryptographic constraints. If it outputs a success indication, go to the next step. Otherwise, set the current status to the returned indication and subcode and go back to step 1.

4. Apply the Chain Constraints to the chain. Certificate meta-data has to be taken into account when checking these constraints against the chain. If the chain does not match these constraints, set the current status to INVALID/CHAIN_CONSTRAINTS_FAILURE and go to step 1.

5. Terminate with the current status and, if VALID, the certificate chain and the calculated control-time returned in step 3.

5.6.1.2 Control-time sliding process

5.6.1.2.1 Description

This process will slide the control-time from the current-time to some date in the past each time it encounters a certificate proven to be revoked.

5.6.1.2.2 Input

Input	Requirement
A prospective certificate chain	Mandatory
A set of POEs	Mandatory
Cryptographic constraints	Optional

5.6.1.2.3 Output

Indication	
VALID + control-time	
INDETERMINATE	*NO_POE*

5.6.1.2.4 Processing

The following steps shall be performed:

1. Initialize control-time to the current date/time.

2. For each certificate in the chain starting from the first certificate (the certificate issued by the trust anchor), do the following:

   a) Find revocation status information satisfying the following:

   - The revocation status information is consistent with the rules conditioning its use to check the revocation status of the considered certificate. For instance, in the case of a CRL, it shall satisfy the checks described in (see [4] clause 6.3).

   - The issuance date of the revocation status information is before control-time.

If more than one revocation status is found, consider the most recent one and go to the next step. If there is no such information, terminate with *INDETERMINATE/NO_POE:*

   b) If the set of POEs contains a proof of existence of the certificate and the revocation status information at (or before) control-time, go to step c). Otherwise, terminate with *INDETERMINATE/NO_POE*.

   c) Update the value of control-time as follows:

   - If the certificate is marked as revoked in the revocation status information, set control-time to the revocation date.

   - If the certificate is not marked as revoked.

     - If the revocation status information is not considered "fresh", set control-time to the issuance date of the revocation status information.

     - Otherwise, the value of control-time is not changed.

   d) Apply the cryptographic constraints to the certificate and the revocation status information. If the certificate (or the revocation status information) does not match these constraints, set *control-time* to the lowest time up to which the listed algorithms were considered reliable.

6) Continue with the next certificate in the chain or, if no further certificate exists, terminate with *VALID* and the calculated control-time.

NOTE 1: In step 1, initializing control-time with current date/time assumes that the trust anchor is still trusted at the current date/time. The algorithm can capture the very exotic case where the trust anchor is broken (or becomes untrusted for any other reason) at a known date by initializing control-time to this date/time.

NOTE 2: The rational of step 2-a) is to check that the revocation status information is "in-scope" for the given certificate. In other words, the rationale is to check that the revocation status information is reliable to be used to ascertain the revocation status of the given certificate. For instance, this includes the fact the certificate is not expired at the issuance date of the revocation status information, unless the issuing CA states that its issues revocation information status for expired certificates (for instance, using the CRL extension expiredCertOnCRL).

NOTE 3: If the certificate (or the revocation status information) was authentic, but the signature has been faked exploiting weaknesses of the algorithms used, this is assumed only to be possible after the date the algorithms are declared to be no longer acceptable. Therefore, the owner of the original key pair is assumed to having been under control of his key up to that date. This is the rational of sliding control-time in step 2-d).

NOTE 4: For more readability, the algorithm above implicitly assumes that the revocation information status is signed by the certificate's issuer which is the most traditional revocation setting but not the only one. The same algorithm can be adapted to the cases where the revocation information status has its own certificate chain by applying the control-time sliding process to this chain which would output a control-time that has to be compared to the control-time associated to the certificate.

5.6.1.3 POE extraction

5.6.1.3.1 Description

This building block derives POEs from a given time-stamp. This process assumes the following about the time-stamp:

- The time-stamp has been accepted as *VALID*.
- The cryptographic hash function used in the time-stamp (*MessageImprint.hashAlgorithm*) is considered reliable at current time or, if this is not the case, a PoE for that timestamp exists for a time when the hash function has still been considered reliable.

In the simple case, a time-stamp gives a POE for each data item protected by the time-stamp at the generation date/time of the token. For instance, a time-stamp on the signature value gives a POE of the signature value (the binary data) at the generation date/time of the time-stamp.

A time-stamp may also give an indirect POE when it is computed on the hash value of some data instead of the data itself. In this case, we will use the following property (indirect POE):

- If we have a POE for h(d) at a date T_1,where h is a cryptographic hash function and d is some data (e.g. a certificate).
- And h is asserted in the cryptographic constraints to be trusted until at least a date T after T_1.
- And we have a POE for d at a date T after T_1.

Then, we can derive from the time-stamp a POE for d at T_1.

5.6.1.3.2 Input

Input	Requirement
Signature	Mandatory
An attribute with a time-stamp token	Mandatory
A set of POEs	Mandatory (but may be empty)
Cryptographic constraints	Optional

5.6.1.3.3 Output

A set of POEs.

5.6.1.3.4 Processing

The following steps shall be performed, depending on the type of the AdES time-stamp.

5.6.1.3.4.1 Extraction from a time-stamp on the signature

Return the set of POEs resulting from the following: add a POE for the signature value at the generation time of the time-stamp.

NOTE: It is possible to infer an indirect POE for the signed data objects (including the signed attributes). However, this is true for some signature algorithms but not all of them (in particular this require that the signature algorithm has the message recovery property and that we have a proof of existence of the public key at the generation time of the time-stamp).

5.6.1.3.4.2 Extraction from a time-stamp on certificates and revocation references

Return the set of POEs resulting from the following. All the POEs are added with the generation time of the time-stamp on certificates and revocation references.

For each reference in the attribute complete-certificate-references and complete-revocation-reference:

1. Add a POE for the hash value h(C) of the certificate C (respectively h(R) of the revocation status information R).
2. If the set of POEs includes a POE for a certificate C (respectively a revocation status information R) at a date/time T after the generation date/time of the time-stamp, add a POE for C (respectively R).

5.6.1.3.4.3 Extraction from a time-stamp on the signature and certificates and revocation references

Return the set of POEs resulting from the following. All the POEs are added with the generation time of the time-stamp on the signature and certificates and revocation references:

1. Do the extraction process from a time-stamp on the signature (see clause 5.6.1.3.4.1).
2. Do the extraction process from a time-stamp on certificates and revocation references (see clause 5.6.1.3.4.2f).

5.6.1.3.4.4 Extraction from an archive-time-stamp

Return the set of POEs resulting from the following. All the POEs are added with the generation time of the archive time-stamp:

1. Add a POE for each signed object.
2. Add a POE for the signature value.
3. Add a POE for each certificate and revocation status information present in the signature.
4. Add a POE for each signed and unsigned attribute (except the attribute containing this archive time-stamp and any archive-time-stamp attribute added after this attribute) present in the signature. This implicitly includes the addition of a POE (direct or indirect POE) for any time-stamp, certificate or revocation information status encapsulated in these attributes.

5.6.1.3.4.5 Extraction from a long-term-validation attribute

This process applies only to CAdES [1]. If the long-term-validation attribute does not include the poeValue field, no POEs are extracted. If the poeValue field is present with a time-stamp, perform the process below. Processing poeValue field when an ERS [17] is present is out of the scope of the present document.

Return the set of POEs resulting from the following. All the POEs are added with the generation time of the time-stamp present in the poeValue:

1. Add a POE for the signed object if available in the SignedData.
2. Add a POE for the signature value.
3. Add a POE for each certificate (respectively revocation information status) in SignedData.certificates (respectively in SignedData.crls) or in long-term-validation.extraCertificates (respectively in long-term-validation.extraRevocation).
4. Add a POE for each signed and unsigned attribute (except the attribute containing this poeValue and the long-term-validation attributes added after it). This implicitly includes the addition of a POE (direct or indirect POE) for any time-stamp, certificate or revocation information status encapsulated in these attributes.

5.6.1.3.4.6 Extraction from a PDF document time-stamp

This process applies only to PAdES [14].

Return the set of POEs resulting from the following. All the POEs are added with the generation time of the document time-stamp:

1. Add a POE for any SignedData included in the ByteRange protected by the document time-stamp. This implicitly includes the addition of a POE (direct or indirect POE) for any time-stamp token, certificate or revocation information status encapsulated in these SignedData.

2. Add a POE for each certificate or revocation information status in a Document Security Store included in the ByteRange protected by the document time-stamp.

3. Add a POE for each document time-stamp included in the ByteRange protected by the document time-stamp. This implicitly includes the addition of a POE (direct or indirect POE) for any certificate or revocation information status encapsulated in these time-stamps.

5.6.1.4 Past signature validation process

5.6.1.4.1 Description

This process is used when validation of a signature (or a time-stamp token) fails at the current time with an *INDETERMINATE* status such that the provided proofs of existence may help to go to a determined status.

5.6.1.4.2 Input

Input	Requirement
Signature	Mandatory
The current time status indication/subcode	Mandatory
Target certificate	Mandatory
X.509 Validation Parameters	Mandatory
A set of POEs	Mandatory
Certificate meta-data	Optional
Chain Constraints	Optional
Cryptographic constraints	Optional

5.6.1.4.3 Output

This process outputs an indication/subcode, which is either the same as the current time indication/subcode given in the inputs or one of the following: *VALID, INVALID/NOT_YET_VALID.*

5.6.1.4.4 Processing

1. Perform the past certificate validation process with the following inputs: the signature, the target certificate, the X.509 validation parameters, certificate meta-data, chain constraints, cryptographic constraints and the set of POEs. If it returns VALID/control-time, go to the next step. Otherwise, return the current time status and subcode with an explanation of the failure.

2. If there is a POE of the signature value at (or before) control-time do the following:

   - If current time indication/subcode is INDETERMINATE/REVOKED_NO_POE or INDETERMINATE/ REVOKED_CA_NO_POE, return VALID.

   - If current time indication/subcode is INDETERMINATE/OUT_OF_BOUNDS_NO_POE: say best-signature-time is the lowest time at which there exists a POE for the signature value in the set of POEs:

     a) If best-signature-time is before the issuance date of the signer's certificate (*notBefore* field), terminate with *INVALID/NOT_YET_VALID.*

     b) If best-signature-time is after the issuance date and before the expiration date of the signer's certificate, return VALID.

   - If current time indication/subcode is *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_*POE and for each algorithm (or key size) in the list concerned by the failure, there is a POE for the material that uses this algorithm (or key size) at a time before to the time up to which the algorithm in question was considered secure, return *VALID.*

In all other cases, return current time indication/subcode together with an explanation of the failure.

5.6.2 Long Term Validation Process

5.6.2.1 Description

An AdES-A (Archival Electronic Signature) is built on an XL signature (EXtended Long Electronic Signature). Several unsigned attributes may be present in such signatures:

- Time-stamp(s) on the signature value (AdES-T).
- Attributes with references of validation data (AdES-C).
- Time-stamp(s) on the references of validation data (AdES-XT2).
- Time-stamp(s) on the references of validation data, the signature value and the signature time-stamp (AdES-XT1).
- Attributes with the values of validation data (AdES-XL).
- Archive time-stamp(s) on the whole signature except the last archive time-stamp (AdES-A).

The process described in this clause is able to validate any of the forms above but also any basic form (namely BES and EPES).

The process handles the AdES signature as a succession of layers of signatures. Starting from the most external layer (e.g. the last archive-time-stamp) to the most inner layer (the signature value to validate), the process performs the basic signature validation algorithm (see clause 8 for the signature itself and clause 7 for the time-stamps). If the basic validation outputs *INDETERMINATE/REVOKED_NO_POE*, *INDETERMINATE/OUT_OF_BOUNDS_NO_POE* or *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE*, we perform the past certificate validation which will output a control-time in the past. The layer is accepted as *VALID*, provided we have a proof of existence before this control-time.

The process does not necessarily fail when an intermediate time-stamp gives the status *INVALID or INDETERMINATE* unless some validation constraints force the process to do so. If the validity of the signature can be ascertained despite some time-stamps which were ignored due to *INVALID* (or *INDETERLINATE*) status, the SVA shall report this information to the DA. What the DA does with this information is out of the scope of the present document.

5.6.2.2 Input

Input	Requirement
Signature	Mandatory
Signed data object (s)	Optional
Trusted-status Service Lists	Optional
Signature Validation Policies	Optional
Local configuration	Optional
A set of POEs	Optional
Signer's Certificate	Optional

5.6.2.3 Output

The main output of this signature validation process is a status indicating the validity of the signature. This status may be accompanied by additional information (see clause 4).

5.6.2.4 Processing

The following steps shall be performed:

1. POE initialization: Add a POE for each object in the signature at the current time to the set of POEs.

NOTE 1: The set of POE in the input may have been initialized from external sources (e.g. provided from an external archiving system). These POEs will be used without additional processing.

2. Basic signature validation: Perform the validation process for AdES-T signatures (see clause 9) with all the inputs, including the processing of any signed attributes/properties as specified.

   - If the validation outputs *VALID*

     - If there is no validation constraint mandating the validation of the LTV attributes/properties, terminate with the indication VALID.

     - Otherwise, go to step 3.

   - If the validation outputs one of the following: INDETERMINATE/REVOKED_NO_POE, INDETERMINATE/REVOKED_CA_NO_POE, INDETERMINATE/OUT_OF_BOUNDS_NO_POE or INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE, go to the next step.

   - In all other cases, fail with returned code and information.

NOTE 2: We go to the LTV part of the validation process in the cases INDETERMINATE/REVOKED_NO_POE, INDETERMINATE/REVOKED_CA_NO_POE, INDETERMINATE/OUT_OF_BOUNDS_NO_POE and INDETERMINATE/ CRYPTO_CONSTRAINTS_FAILURE_NO_POE because additional proof of existences may help to go from INDETERMINATE to a determined status.

NOTE 3: Performing the LTV part of the algorithm even when the basic validation gives VALID may be useful in the case the SVA is controlled by an archiving service. In such cases, it may be necessary to ensure that any LTV attribute/property present in the signature is actually valid before making a decision about the archival of the signature.

NOTE 4: Steps 3 to 7 below are not part of the validation process per se, but are present to collect PoEs for step 8.

3. If there is at least one long-term-validation attribute with a poeValue, process them, starting from the last (the newest) one as follows: Perform the time-stamp validation process (see clause 8) for the time-stamp in the poeValue:

   a) If *VALID* is returned and the cryptographic hash function used in the time-stamp (*MessageImprint.hashAlgorithm*) is considered reliable at the generation time of the time-stamp: Perform the POE extraction process with the signature, the long-term-validation attribute, the set of POEs and the cryptographic constraints as inputs. Add the returned POEs to the set of POEs.

   b) Otherwise, perform past signature validation process with the following inputs: the time-stamp in the poeValue, the status/subcode returned in step 3a, the TSA's certificate, the X.509 validation parameters, certificate meta-data, chain constraints, cryptographic constraints and the set of POEs. If it returns VALID and the cryptographic hash function used in the time-stamp is considered reliable at the generation time of the time-stamp, perform the POE extraction process and add the returned POEs to the set of POEs. In all other cases:

      - If no specific constraints mandating the validity of the attribute are specified in the validation constraints, ignore the attribute and consider the next long-term-validation attribute.

      - Otherwise, fail with the returned indication/subcode and associated explanations

4. If there is at least one archive-time-stamp attribute, process them, starting from the last (the newest) one, as follows: perform the time-stamp validation process (see clause 8):

   a) If *VALID* is returned and the cryptographic hash function used in the time-stamp (*MessageImprint.hashAlgorithm*) is considered reliable at the generation time of the time-stamp: Perform the POE extraction process with the signature, the archive-time-stamp, the set of POEs and the cryptographic constraints as inputs. Add the returned POEs to the set of POEs.

   b) Otherwise, perform past signature validation process with the following inputs: the archive time-stamp, the status/subcode returned in step 4a, the TSA's certificate, the X.509 validation parameters, certificate meta-data, chain constraints, cryptographic constraints and the set of POEs. If it returns VALID and the cryptographic hash function used in the time-stamp is considered reliable at the generation time of the

time-stamp, perform the POE extraction process and add the returned POEs to the set of POEs. In all other cases:

- If no specific constraints mandating the validity of the attribute are specified in the validation constraints, ignore the attribute and consider the next archive-time-stamp attribute.
- Otherwise, fail with the returned indication/subcode and associated explanations.

NOTE 4: If the signature is PAdES, document time-stamps replace archive-time-stamp attributes and the process "Extraction from a PDF document time-stamp" replaces the process "Extraction from an archive-time-stamp".

5. If there is at least one time-stamp attribute on the references, process them, starting from the last one (the newest), as follows: perform the time-stamp validation process (see clause 8):

   a) If *VALID* is returned and the cryptographic hash function used in the time-stamp (*MessageImprint.hashAlgorithm*) is considered reliable at the generation time of the time-stamp, perform the POE extraction process with the signature, the time-stamp on the references, the set of POEs and the cryptographic constraints. Add the returned POEs to the set of POEs.

   b) Otherwise, perform past signature validation process with the following inputs: the time-stamp on the references, the status/subcode returned in step 5a, the TSA's certificate, the X.509 validation parameters, certificate meta-data, chain constraints, cryptographic constraints and the set of POEs:

      - If it returns VALID and the cryptographic hash function used in the time-stamp is considered reliable at the generation time of the time-stamp, perform the POE extraction process and add the returned POEs to the set of POEs. In all other cases:
      - If no specific constraints mandating the validity of the attribute are specified in the validation constraints, ignore the attribute and consider the next archive-time-stamp attribute.

Otherwise, fail with the returned indication/subcode and associated explanations.

6. If there is at least one time-stamp attribute on the references and the signature value, process them, starting from the last one, as follows: perform the time-stamp validation process (see clause 8):

   a) If *VALID* is returned and the cryptographic hash function used in the time-stamp (*MessageImprint.hashAlgorithm*) is considered reliable at the generation time of the time-stamp, perform the POE extraction process with the signature, the time-stamp, the set of POE and the cryptographic constraints. Add the returned POEs to the set of POEs.

   b) Otherwise, perform past signature validation process with the following inputs: the time-stamp, the status/subcode returned in step 6a, the TSA's certificate, the X.509 validation parameters, certificate meta-data, chain constraints, cryptographic constraints and the set of POEs. If it returns VALID and the cryptographic hash function used in the time-stamp is considered reliable at the generation time of the time-stamp, perform the POE extraction process and add the returned POEs to the set of POEs. In all other cases:

      - If no specific constraints mandating the validity of the attribute are specified in the validation constraints, ignore the attribute and consider the next archive-time-stamp attribute.
      - Otherwise, fail with the returned indication/subcode and associated explanations:

7. If there is at least one signature-time-stamp attribute, process them, in the order of their appearance starting from the last one, as follows: Perform the time-stamp validation process (see clause 8)

   a) If *VALID* is returned and the cryptographic hash function used in the time-stamp is considered reliable at the generation time of the time-stamp, perform the POE extraction process with the signature, the signature-time-stamp, the set of POEs and the cryptographic constraints. Add the returned POEs to the set of POEs.

   b) Otherwise, perform past signature validation process with the following inputs: the time-stamp, the status/subcode returned in step 7a, the TSA's certificate, the X.509 validation parameters, certificate meta-data, chain constraints, cryptographic constraints and the set of POEs. If it returns VALID and the cryptographic hash function used in the time-stamp (*MessageImprint.hashAlgorithm*) is considered

reliable at the generation time of the time-stamp, perform the POE extraction process and add the returned POEs to the set of POEs. In all other cases:

- If no specific constraints mandating the validity of the attribute are specified in the validation constraints, ignore the attribute and consider the next archive-time-stamp attribute.
- Otherwise, fail with the returned indication/subcode and associated explanations

8. Past signature validation: perform the past signature validation process with the following inputs: the signature, the status indication/subcode returned in step 2, the signer's certificate, the X.509 validation parameters, certificate meta-data, chain constraints, cryptographic constraints and the set of POEs. If it returns VALID go to the next step. Otherwise, abort with the returned indication/subcode and associated explanations.

Data extraction: the SVA shall return the success indication *VALID*. In addition, the SVA **should** return additional information extracted from the signature and/or used by the intermediate steps. In particular, the SVA should return intermediate results such as the validation results of any time-stamp token or time-mark. What the DA does with this information is out of the scope of the present document.

Annex A Annex A (informative): Validation Constraints

Any requirements in this clause are extracted from other documentation. No new requirement is introduced in the present document. The details of how to validate such constraints will not be given in the present document. Such constraints are listed only to give a complete overview of all constraints that are considered important for the purpose of the present document. It also is not intended as a complete list of constraints a SVA may need to consider.

The use of any of the constraints may however be forced to be ignored by the SVA, depending on the signature validation policy in force.

A.1 X.509 Certificate validation constraints

The following constraints are provided for use in the certification path validation process as defined in RFC 5280 [4]. Constraints defined in the tables below may be different for different certificate types (end-entity signer's certificates, time-stamp signing authority certificates, CA certificates, etc.)

Table A.1

Constraint	Description	Reference
A set of trust anchor information	The DA provides the SVA a list of acceptable trust anchors as a constraint for the validation process. Such TAs are recommended to be provided in the form of (self-signed) certificates and a time until when these trust anchors were considered reliable. The TA information may be taken from: • Trust points specified in signature validation policies • Sets of trusted CAs, e.g. represented by their root certificates stored in the environment (like certificate trust store or list) • Trust Service Status Lists as defined in [3] • Trusted Lists as defined in [i.6] The DA may also provide the TA information to the SVA in one of these forms, if applicable.	[4], [i.1], CD 2009/767/EC [i.6] amended by CD 2010/425/EU [i.3], [i.2]
A certification path	This constraint consists in the provision of a certification path of	[4]

	length 'n' from the TA down to the certificate used in creating a signed object (e.g. the signer's certificate or a time-stamping certificate). The given certification path has to be used by the SVA for validation of the signature.	
`user-initial-policy-set`	*"A set of certificate policy identifiers naming the policies that are acceptable to the DA. The user-initial-policy-set contains the special value any-policy when not concerned about certificate policy".*	[4]
`initial-policy-mapping-inhibit`	*"Indicates if policy mapping is allowed in the certification path".*	[4]
`initial-explicit-policy`	*"Indicates if the path must be valid for at least one of the certificate policies in the user-initial-policy-set".*	[4]
`initial-any-policy-inhibit`	*"Indicates whether the anyPolicy OID should be processed if it is included in a certificate".*	[4]
`initial-permitted-subtrees`	*"Indicates for each name type (e.g. X.500 distinguished names, email addresses, or IP addresses) a set of subtrees within which all subject names in every certificate in the certification path MUST fall".*	[4]
`initial-excluded-subtrees`	*"Indicates for each name type (e.g. X.500 distinguished names, email addresses, or IP addresses) a set of subtrees within which no subject name in any certificate in the certification path may fall".*	[4]

Additional Chain Constraints:

The following types of constraints will be applied in the XCV building block. Some of the constraints may be intrinsically defined by a CA using extensions in the certificates themselves, like NameConstraints etc. SVAs are assumed to handle such constraints as defined in the relevant. The DA may need to define initial values for these constraints or want the SVA to handle such constraints differently (e.g. ignore them).

Table A.2

Constraint	Description	X.509-extension	Reference
Path-Length Constraints	Restrictions on the number of CA certificates in a certification path.	`BasicConstraints`	[4], [i.1], [i.2], [i.3]
Policy Constraints	Defines constraints for certificate policies referenced in the certificates.	`PolicyConstraints`	[4], [i.1], [i.2], [i.3]
Name Constraints	Defines constraints on the distinguished names (DN) for issued certificates.	`NameConstraints`	[4], [i.1], [i.2], [i.3]

Additional Revocation Constraints:

The following constraints will be applied when verifying the certificate validity status of the certificates during the certification path validation process.

Table A.3

Constraint	Description	Reference
Revocation Checking Constraints	Indicates requirements for checking certificate revocation. Such constraints may specify: • If revocation checking is required or not • If OCSP responses or CRLs have to be used One possible syntax/semantic for a set of requirement values used to express such requirements is defined in TR 102 272 [i.2] and TR 102 038 [i.3]: **"*clrCheck:*** *Checks shall be made against current CRLs (or ARLs);* ***ocspCheck:*** *The revocation status shall be checked using OCSP RFC 2560 [i.9];* ***bothCheck:*** *Both OCSP and CRL checks shall be carried out;*	[i.2], [i.3]

	eitherCheck: *Either OCSP or CRL checks shall be carried out;* ***noCheck:*** *No check is mandated."*	
Revocation Freshness Constraints	Used to time constraints on revocation information. The constraints may indicate the maximum accepted difference between the issuance date of the revocation status information of a certificate and the time of validation (see clause 4.5) or require the SVA to only accept revocation information issued a certain time after the signature has been created.	present document, clause 4.4
Revocation Info of expired certificates	This constraint mandates the signer's certificate used in validating the signature to be issued by a certificate authority that keeps revocation notices for revoked certificates even after they have expired for a period exceeding a given lower bound (see note).	[8], [6]
NOTE: The Revocation Info of expired certificates-constraint may be more efficiently implementable by not including such a CA in the list of trust anchors.		

Additional Time-Stamp Trust Constraints:

The following constraints will be applied, when applicable, on the time-stamp present in a signature:

Table A.4

Constraint	Description	Reference
`TimestampDelay`	Indicates a maximum acceptable delay between the signing time as claimed by the signer and the time included within the signature Timestamp (i.e. AdES-T).	[i.2], [i.3]

The following constraints are to be applied to the signer's certificate before considering it as valid for the intended use.

Table A.5

Constraint	Description	Reference
`QualifiedCertificate`	Mandates the signer's certificate used in validating the signature to be a qualified certificate as defined in Directive 1999/93/EC [i.15].How to derive this status from certificate is further detailed in Annex B.2	[5], [7], CD 2009/767/EC [i.6] amended by CD 2010/425/EU DTS-ESI-000099,B.3,(h)
`SSCD`	Mandates the end user certificate used in validating the signature to be supported by a secure signature creation device (SSCD) as defined in Directive 1999/93/EC [9]. How to derive this status from certificate is further detailed in Annex B.2. •	[i.15], [7], CD 2009/767/EC [i.6] amended by CD 2010/425/EU SR 001 604 [i.10], clause B.3 (n)
`ForLegalPerson`	Mandates the signer's certificate used in validating the signature to be issued by a certificate authority issuing certificate as having been issued to a legal person.	CD 2009/767/EC [i.6] amended by CD 2010/425/EU SR 001 604 [i.10], clause B.3,(l)

A.2 Cryptographic Constraints

Cryptographic constraints are applied on algorithms and parameters used when validating signed objects included in the validation process (e.g. signature, certificates, CRLs, OCSP responses, time-stamps). They will typically be represented by a list of entries, each consisting of:

- An identifier for the algorithm.
- The type of signature to which the constraint applies (e.g. signature to be validated, signer's certificate, CA certificates in a valid chain, TST signature, OCSP response signature, CRL signature).
- For signature algorithms: The minimum key size.
- For hash algorithms: The minimum length of the hash value, if the hash function allows for hash values of different size.
- An expiration date: This date specifies, until when the given algorithm/key size or algorithm/hash length combination is accepted as being strong enough.

NOTE: The expiration date is necessary to be able to check signatures in the past. An algorithm, like RSA, may therefore appear more than once in the list, since the acceptable key size will change with time.

A.3 Constraints on Signature Elements

Table A.6

Constraint	Description	Reference
SigningCertificate chain constraint	If the signature includes a specific chain in the SigningCertificate signed property, it is mandated to be part of the validated certification paths.	[1], [2], [12]
MandatedSignedQProperties	Indicates the mandated signed qualifying properties that are mandated to be present in the signature. This includes: • signing-time • data-object-format • content-hints • content-reference • content-identifier • commitment-type-indication • signer-location / signature-production-place • signer-attributes / signer-role • content-time-stamp	[i.3] SR 001 604 [i.10], B.3,(a), (e), (i), (o)
MandatedUnsignedQProperties	Indicates the mandated unsigned qualifying properties that are mandated to be present in the signature. This constraint may be applicable to either the signer or the verifier. This includes: • counter-signature • mandated signature time-stamp (i.e. AdES-T) • mandated LT form • mandated archival form (-A) • signature policy extensions	[i.3] SR 001 604 [i.10], B.3,(k)
Constraints on Roles	This includes: • RoleMandated • HowCertRoles • RoleType constraints • RoleValue constraints • Role constraints	[i.3] SR 001 604 [i.10], B.3,(m)

Annex B (informative): Certificate Meta-Data

This annex lists types of certificate meta-data that the DA may make available to the SVA. This is data that is required to check constraints which are e.g. part of a signature validation policy but is not or not easily available to the SVA. Making such meta-data available to the SVA will therefore result more often in a VALID or INVALID response, where the SVA would need to return INDETERMINATE should that information not be available.

NOTE: While some of this meta-data may be retrieved form a Trust-service Status List (TSL) [3] or a Trusted List [i.16], the same type of information may be available to the DA in other forms, but are semantically equivalent.

Table B.1

Meta-data	Description	Reference
QcStatements	Declares that a certificate qualified status can be recognized by checking the QCStatements-extension.	[5]
QCP(+)	Declares that a certificate has been issued under a QCP(+) policy as defined in [7].	[7]
NCP(+), LCP	Declares that a certificate has been issued under a NCP(+) or a LCP policy, resp., as defined in [8].	[8]
QCWithSSCD	Declares that when a certificate has been issued as a qualified certificate the private key associated with the public key in the certificate resides within a Secure Signature Creation Device.	[3]
QCNoSSCD	Declares that when a certificate has been issued as a qualified certificate the private key associated with the public key in the certificate does not reside within a Secure Signature Creation Device.	[3]
QCForLegalPerson	Declares that when a certificate has been issued as a qualified certificate it has been issued to a legal person.	[3]
WithSSCD	Declares that the private key associated with the public key in a certificate resides within a Secure Signature Creation Device.	
NoSSCD	Declares that the private key associated with the public key in a certificate does not reside within a Secure Signature Creation Device.	
ForLegalPerson	Declares that a certificate has been issued to a legal person.	
expiredCertsRevocationInfo	Declares that a CRL or OCSP issuer issues CRL and/or OCSP responses that keep revocation notices for revoked certificates also after they have expired.	[3], [6]

B.1 Deriving "qualified" status from a certificate

The "qualified" status of a certificate in the sense of Directive 1999/93/EC [i.15] can be derived from:

- QcCompliance extension being set in the signer's certificate in accordance with TS 101 862 [5];
- QCP+ or QCP certificate policy OID being indicated in the signer's certificate policies extension (i.e. 0.4.0.1456.1.1 or 0.4.0.1456.1.2);
- The content of a Trusted List through information provided in the Sie field of the applicable service entry; or
- Static configuration that provides such information in a trusted manner.

A certificate may contain a claim to be qualified, or may not contain such a claim. Per CD 2009/767/EC [i.6] the Trusted List of the Member State in which the issuer of the certificate is established is decisive for determining the status: It may contain a Qualifications Service information extension [i.16] that declares a certificate to be qualified, even if the certificate does not contain that claim, or that a certificate is not qualified, even if the certificate contains that claim.

B.2 Deriving "supported by an SSCD" status from a certificate

The "supported by an SSCD" status of a certificate in the sense of Directive 1999/93/EC [i.15] can be derived from:

- QcSSCD extension being set in the signer's certificate in accordance with TS 101 862 [5];
- QCP+ certificate policy OID being indicated in the signer's certificate policies extension (i.e. 0.4.0.1456.1.1);
- The content of a Trusted List through information provided in the appropriate extension field of the applicable service entry; or
- Static configuration that provides such information in a trusted manner.

A certificate may contain a claim that corresponding private key resides in an SSCD, or may not contain such a claim. Per CD 2009/767/EC [i.6] the Trusted List of the Member State in which the issuer of the certificate is established is decisive for determining the status: It may contain a Qualifications Service information extension [i.16] that declares the private key corresponding to the certificate to reside in an SSCD, even if the certificate does not contain such a claim, or that it the private key in fact does not reside in an SSCD, if the certificate contains that claim.

Annex C (informative): Validation Examples

This clause gives some examples that aim at helping to better understand the signature validation algorithm presented in the normative part of the present document. To achieve this goal, we run through the document step by step only for the critical elements of the algorithm.

C.1 General remarks and assumptions

- While validating an AdDS-T signature is specified in a separate clause (see clause 5.5), this has been done only to keep this special case simple. It would have been perfectly possible to use the LTV/algorithm also for the T-form. In the examples we ignore this distinction and only present the logic behind the algorithm as applicable to the examples chosen.
- These examples also assume that basic checks like cryptographic or format checks succeed. We concentrate on examples showing how the fundamental properties of an AdES signature, proving the existence of certain objects at certain times, help to validate signatures from the past.
- For all validation examples, we assume to be able to identify the signer's certificate, as it is provided within the signature.
- We assume not to have any specific constraints on the validation process unless noted otherwise.
- We assume that a valid path to a trust anchor can be built for all certificates used unless noted otherwise.
- We assume only to have the signature as an input unless noted otherwise.
- We assume that the syntax/format of all elements is ok, that all required elements are there, that time-stamps and signatures have been calculated over the right data and no other similar basic flaws exist, unless noted otherwise.

C.2 Symbols

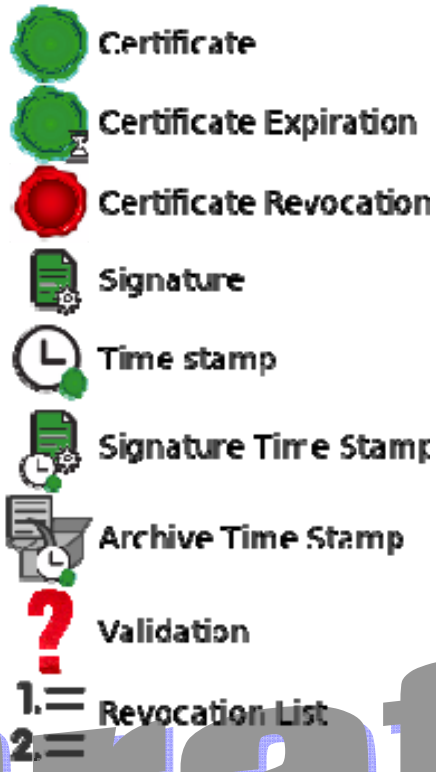


Figure C.1: Symbols used in examples

Figure C.1 shows the symbols used in the following graphics.

C.3 Example 1: Revoked certificate

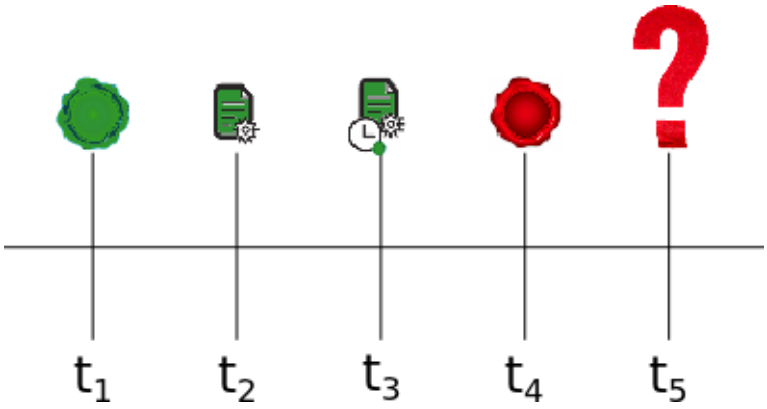


Figure C.2: Revoked Certificate Example

In this example we look at a simple case where a certificate is revoked before subsequent validation of a signature. Figure C.2 shows the timeline for the relevant events:

- At time t1 the certificate is issued.
- At time t2 the signature is created using the certificate.
- At time t3 a signature timestamp is created.
- At time t4 the certificate is revoked.

- At time t5 we try to validate the certificate.
- All other certificates that are used in the process are assumed to being still valid.

Let us try to go through the steps involved in different signature validation scenarios for this example.

C.3.1 AdES-BES/EPES

Expected result	*INDETERMINATE*/REVOKED_NO_POE
Rational	The BES validation algorithm does not process the signature-time-stamp attribute and hence cannot ascertain whether the signing time is before the revocation date. Hence, the validity status is indeterminate.

Let us try to use the validation algorithm defined in clause 6:

- Identify the signer's certificate: succeeds by assumption.
- Initialize the validation constraints and parameters: Succeeds by assumption.
- Validate the signer's certificate: will return *INDETERMINATE* / *REVOKED_NO_POE* since the signer's certificate has been revoked.

The algorithm terminates with *INDETERMINATE*/*REVOKED_NO_POE* which is expected and correct.

C.3.2 AdES-T

Expected result	*VALID*
Rational	The status goes from *INDETERMINATE/REVOKED_NO_POE* (using the AdES-BES validation algorithm) to *VALID* because the AdES-T validation algorithm will process the signature time-stamp attribute and will find that the signing time lies before the revocation date.

Let us try to use the AdES-T-validation algorithm defined in clause 8:

- We initialize the set of signature time-stamp tokens to the single time-stamp present in the signature (step 1).
- Best-signature-time is set to current time (step 1).
- *Signature validation: Perform the validation process for BES signatures (step 2).* As we have seen before, this returns *INDETERMINATE/REVOKED_NO_POE*, and we proceed with the rest of the algorithm, since we hope (or know) that existing time-stamps may still allow us to verify the signature.
- Verification of time-marks (step 3). No time-marks by assumption.
- *Message imprint verification (step 4-a)*: we check the message imprint of the time-stamp, which succeeds by assumption.
- *Time-stamp token validation (step 4-b)*: we now move to clause 7 for verifying the time-stamp.
- We perform BES-validation of the signature on the time-stamp token, which succeeds, since we assume that the certificate of the TSA has neither expired nor been revoked.
- Since the previous step returned VALID, we now can assume the signature has been created before the timestamp we can set best-signature-time to the time of the timestamp (step 4-b).
- Step 5-a compares this best signature time with the revocation date of the certificate. Since the certificate has been revoked only after the time-stamp has been generated, we can continue.
- The coherence of the time values is checked and found to be ok (step 5-c).
- We have no constraints on time-stamp delay (step 6), so we skip the next step.
- We now can return VALID and return the validation report generated to the DA (step 7).

C.4 Example 2: Revoked CA certificate

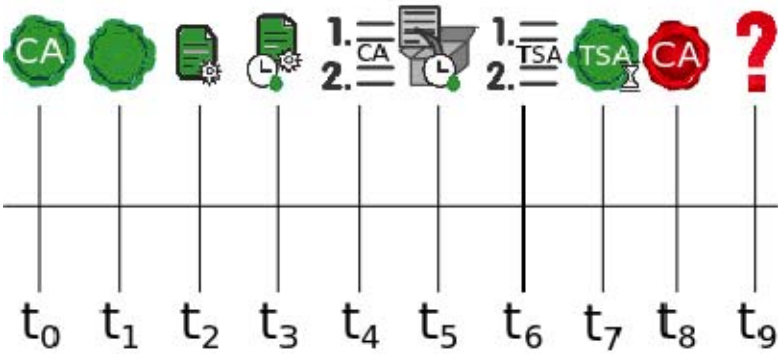


Figure C.3: Revoked CA Certificate

Next we look at a slightly more complex case, where the CA certificate that issued the signers certificate has been revoked. Figure C.3 shows the timeline for the relevant events:

- At time t0 the CA certificate is issued by another CA.
- At time t1 the signers certificate is issued by that CA.
- At time t2 the signature is created using the certificate.
- At time t3 a signature timestamp is created.
- At time t4 CRLs were issued by the CA that issued the signers certificate.
- At time t5 an AdES-A is created and an archive timestamp produced.
- At time t6 CRLs were issued for the certificate of the Time-Stamping Authority that issued the signature time-stamp.
- At time t7 the certificate of the Time-Stamping Authority that issued the signature time-stamp expires.
- At time t8 the CA certificate is revoked.
- At time t9 we try to validate the certificate.
- All other certificates that are used in the process are assumed to being still valid.

We assume here that the TSA certificate has been issued by a different authority than the CA certificate. Let us try to go through the steps involved in different signature validation scenarios for this example.

C.4.1 AdES-BES/EPES

Expected result	*INDETERMINATE/REVOKED_CA_NO_POE*
Rational	AdES-BES algorithm does not handle the LTV attributes.

Let us try to use the validation algorithm defined in clause 6:

- Identify the signer's certificate: succeeds by assumption.
- Initialize the validation constraints and parameters: Succeeds by assumption.
- Validate the signer's certificate: will return INDETERMINATE/REVOKED_CA because the CA certificate has been revoked.

The algorithm terminates here with INDETERMINATE/REVOKED_CA_*NO_POE*, which is expected and correct.

C.4.2 AdES-T

Expected result	*INDETERMINATE/REVOKED_CA_NO_POE*
Rational	AdES-T algorithm does not handle the LTV attributes. The signature-time-stamp attribute protects only the signature value and the signing certificate but does not help when an intermediary CA is revoked.

Let us try to use the AdES-T-validation algorithm defined in clause 8:

- We initialize the set of signature time-stamp tokens to the single signature time-stamp token present in the signature.
- Best-signature-time is set to current time.
- *Signature validation: Perform the validation process for BES signatures.* As we have seen before, this returns *INDETERMINATE/REVOKED_CA_NO_POE.*
- Since the signature validation did not report VALID nor INDETERMINATE/REVOKED_NO_POE nor INDETERMINATE/OUT_OF_BOUNDS, the algorithm terminates with INDETERMINATE/REVOKED_CA_*NO_POE.*

C.4.3 LTV

Finally, let us do the same process using the LTV-Algorithm.

Expected result	*VALID*
Rational	*INDETERMINATE* turns into *VALID* due to the archive time-stamp which was produced at T5 before any compromising event.

We start in clause 9.2.4:

- *POE initialization (step 1)*: we initialize the POE with all objects we have:

Content	Exists at time
The signature	T9
The Signers Certificate (and other certificates required to form a chain to a trust anchor)	T9
Revocation Information for the Signers Certificate (as well as for all certificates required to form a chain to a trust anchor)	T9
The signature time-stamp	T9
The TSA Certificate related to the signature time-stamp (and other certificates required to form a chain to a trust anchor)	T9
Revocation Information for that TSA Certificate (as well as for all certificates required to form a chain to a trust anchor)	T9
The archive time-stamp	T9
The TSA Certificate related to the archive time-stamp (and other certificates required to form a chain to a trust anchor)	T9
Revocation Information for that TSA Certificate (as well as for all certificates required to form a chain to a trust anchor)	T9

- *Perform the validation process for AdES-T signatures*: As we have seen before, this returns INDETERMINATE/REVOKED_CA_*NO_POE*, and we proceed with the algorithm, since we hope (or know) that existing time-stamps may still allow us to verify the signature.
- *Archive Timestamp Validation (step 4):* We move to clause 7 for verifying the archive time-stamp:
  - We perform BES-validation of the signature on the archive time-stamp token, which succeeds, since we assume that the certificate of the archive-TSA has neither expired nor been revoked.

- we can extract POEs at the time of the archive timestamp (see clause 9.2.3.4.4) for:
  - The signature
  - The Signers Certificate (and other certificates required to form a chain to a trust anchor)
  - Revocation Information for the Signers Certificate (as well as for all certificates required to form a chain to a trust anchor)
  - The signature time-stamp
  - The TSA Certificate related to the signature time-stamp (and other certificates required to form a chain to a trust anchor)

Resulting in the following set of POEs:

Content	Exists at time
The signature	T5
The Signers Certificate (and other certificates required to form a chain to a trust anchor)	T5
Revocation Information for the Signers Certificate (as well as for all certificates required to form a chain to a trust anchor)	T5
The signature time-stamp	T5
The TSA Certificate related to the signature time-stamp (and other certificates required to form a chain to a trust anchor)	T5
Revocation Information for that TSA Certificate (as well as for all certificates required to form a chain to a trust anchor)	T5
The archive time-stamp	T9
The TSA Certificate related to the archive time-stamp (and other certificates required to form a chain to a trust anchor)	T9
Revocation Information for that TSA Certificate (as well as for all certificates required to form a chain to a trust anchor)	T9

- Steps 5 and 6 are skipped, there are no such time-stamps in the signature.
- Step 7: process the signature time-stamp:
  We do the time-stamp validation process (clause 7):
  - We perform BES-validation of the signature on the time-stamp token, which returns INDETERMINATE/OUT_OF_BOUNDS_*NO_POE*, since the certificate of that TSA has expired.
- Since this step returned INDETERMINATE/OUT_OF_BOUNDS_NO_POE, we perform the past signature validation process for the time-stamp (see clause 9.2.4):
  - We perform the past certificate validation for the TSA certificate:
    - The prospective chain can be built (we have all information in the archive).
    - Since the TSA-certificate has only expired, path validation at a point in time, where the TSA-certificate was not yet expired will succeed.
    - We perform the control-time sliding process with the following inputs: the prospective chain and the set of POEs.
      - Control-time is *current time.*
      - We can find revocation objects for the TSA-certificate in the set of POE.
      - We have proof of existence of the relevant objects at T5.
      - We assume the revocation object not to be fresh and thus can now set control-time to the time this revocation object has been created (T7).
      - We apply certificate constraints and cryptographic constraints to the chain, which succeed by assumption.
      - We return with VALID and control-time T7.

- Since the current time status is INDETERMINATE/OUT_OF_BOUNDS_NO_POE and we have a POE for the signature time-stamp at T5 before T7, the past signature validation will return *VALID*.

- We now do the POE-extraction process for that time-stamp and get a new list of POEs.

Content	Exists at time
The signature	T3
The Signers Certificate (and other certificates required to form a chain to a trust anchor)	T3
Revocation Information for the Signers Certificate (as well as for all certificates required to form a chain to a trust anchor)	T4
The signature time-stamp	T5
The TSA Certificate related to the signature time-stamp (and other certificates required to form a chain to a trust anchor)	T5
Revocation Information for that TSA Certificate (as well as for all certificates required to form a chain to a trust anchor)	T5
The archive time-stamp	T9
The TSA Certificate related to the archive time-stamp (and other certificates required to form a chain to a trust anchor)	T9
Revocation Information for that TSA Certificate (as well as for all certificates required to form a chain to a trust anchor)	T9

- We now do the past signature validation process for the signature:
  - We perform the past certificate validation for the signer's certificate:
    - Certificate chain can be built by assumption.
    - Certificate path validation succeeds.
    - We perform the control-time sliding process for the signer's certificate:
      - Control-time is current time.
      - We have a POE at the current time for the CA certificate and the corresponding revocation info status.
      - Since the CA is revoked at t8, control-time takes this value (assuming that freshness does not apply).
      - We have proof of existence of the relevant objects for the signer's certificate at T3 before T8.
      - We assume the revocation object to be fresh and thus do not change control-time.
      - We apply certificate constraints and cryptographic constraints to the chain, which succeeds by assumption.
      - We return with VALID and control-time T8.
    - Since the current time status is INDETERMINATE/REVOKED_CA_NO_POE and we have a POE for the signature at T3 before T8, the past signature validation will return *VALID*.

- The validation algorithm returns a final VALID plus the validation report.

Annex D (informative): Validation process versus signature conformance levels

TS 103 171 [18] profiles the use of XAdES signatures for its use in the context of the "Directive 2006/123/EC [i.7] of the European Parliament and of the Council of 12 December 2006 on services in the internal market" (EU Services Directive henceforth) and any applicable context where qualified signatures are used. TS 103 172 [19] (respectively TS 103 173 [20]) does the same for PAdES (respectively for CAdES). These documents define four conformance levels. Namely: ST-Level (Short Term Level), T-Level (Trusted time for signature existence), LT-Level (Long Term Level) and LTA-Level (Long Term with Archive time-stamps). These conformance levels are defined for encompassing the life cycle of electronic signatures and are built on the AdES forms.

One of the motivations behind presenting the validation procedures in three levels (Basic Validation Process, Validation Process for AdES-T and Long Term Validation Process) is that implementations of the SVA that aim to validate only basic conformance levels are not obliged to implement the LTV building blocks which are much more complicated.

Table D.1 proposes a mapping between the validation processes and the conformance levels that are willing to be validated by each of these processes:

- An SVA that implements the Long Term Validation Process (see clause 5.6.2) is willing to validate signatures conformant to any of the conformance levels (ST, T, LT and LTA).
- An SVA that implements the Validation Process for AdES-T (see clause 5.5) is willing to validate signatures conformant to ST, T or LT levels.
- An SVA that implements the Basic Validation Process (see clause 5.3) is willing to validate signatures conformant to ST level.

Table D.1: Mapping between validation process and signature conformance levels

	Basic Validation Process	Validation Process for AdES-T	Long Term Validation Process
ST level	X	X	X
T level		X	X
LT level		X	X
LTA level			X

Annex E Signature Validation Report

Note: This clause will specify a signature validation report structure in a later version of this draft.

6 History

Document history		
V0.1.1	August 2013	First draft containing signature creation
V0.1.2	August 2013	First rework of Validation section
V0.1.8	October 2013	Rework considering comments received so far
V0.1.11	November 2013	Changes decided in the STF F2F worked in.
V0.2.1	November 2013	Mainly editorial changes. Draft for public review.