ETSI TS 119 102-1 V1.0.1 (2015-07)

Electronic Signatures and Infrastructures (ESI);
Procedures for Creation and Validation
of AdES Digital Signatures;
Part 1: Creation and Validation

Reference
RTS/ESI-0019102-1-TS

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

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).

The present document is part 1 of a multi-part deliverable covering Procedures for Creation and Validation of AdES Digital Signatures, as identified below:

Part 1: "Creation and Validation";

Part 2: "Validation Report".

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

The present document aims to meet the general requirements of the international community to provide trust and confidence in electronic transactions, including, amongst other, applicable requirements from Regulation (EU) No 910/2014 [i.15].

1 Scope

The present document specifies procedures for:

- the creation of AdES digital signatures (specified in ETSI TS 119 122-1 [i.2], ETSI TS 119 132-1 [i.4], ETSI TS 119 142-1 [i.6] respectively);
- establishing whether an AdES digital signature is technically valid;

whenever the AdES digital signature is based on public key cryptography and supported by public key certificates. To improve readability of the document, *AdES digital signatures* are meant when the term *signature* is being used.

NOTE: Regulation (EU) No. 910/2014 [i.15] defines the terms electronic signature, advanced electronic signature, electronic seals and advanced electronic seal. These signatures and seals are usually created using digital signature technology. The present document aims at supporting the Regulation (EU) No 910/2014 [i.15] for creation and validation of advanced electronic signatures and seals when they are implemented as AdES digital signatures.

The present document introduces general principles, objects and functions relevant when creating or validating signatures based on signature creation and validation constraints and defines general classes of signatures that allow for verifiability over long periods.

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; and
- the legal interpretation of any signature, especially the legal validity of a signature.

2 References

2.1 Normative 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.

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

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

[2] ISO/IEC 9594-8:2014: "Information technology -- Open Systems Interconnection -- The Directory -- Part 8: Public-key and attribute certificate frameworks".

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

[4] ETSI TS 119 172-1: "Electronic Signatures and Infrastructures (ESI); Signature Policies; Part 1: Building blocks and table of contents for human readable signature policy documents".

2.2 Informative 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.

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

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 TS 119 122-1: "Electronic Signatures and Infrastructures (ESI); CAdES digital signatures; Part 1: Building blocks and CAdES baseline signatures".

[i.3] ETSI TS 119 122-2: "Electronic Signatures and Infrastructures (ESI); CAdES digital signatures; Part 2: Extended CAdES signatures".

[i.4] ETSI TS 119 132-1: "Electronic Signatures and Infrastructures (ESI); XAdES digital signatures; Part 1: Building blocks and XAdES baseline signatures".

[i.5] ETSI TS 119 132-2: "Electronic Signatures and Infrastructures (ESI); XAdES digital signatures; Part 2: Extended XAdES signatures".

[i.6] ETSI TS 119 142-1: "Electronic Signatures and Infrastructures (ESI); PAdES digital signatures; Part 1: Building blocks and PAdES baseline signatures".

[i.7] ETSI TS 119 142-2: "Electronic Signatures and Infrastructures (ESI); PAdES digital signatures; Part 2: Additional PAdES signatures profiles".

[i.8] IETF RFC 3852: "Cryptographic Message Syntax (CMS)".

[i.9] IETF RFC 4998: "Evidence Record Syntax (ERS)".

[i.10] IETF RFC 6283: "Extensible Markup Language Evidence Record Syntax (XMLERS)".

[i.11] W3C Recommendation (2008): "XML Signature Syntax and Processing".

[i.12] IETF RFC 6960: "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP".

[i.13] ETSI TS 119 422: "Electronic Signatures and Infrastructures (ESI); Time-stamping protocol and time-stamp profiles".

[i.14] ECRYPT II Yearly Report on Algorithms and Keysizes (2010-2011), Revision 1.0, 30. June 2011.

[i.15] Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC.

3 Definitions and abbreviations

3.1 Definitions

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

attribute authority: authority which assigns privileges by issuing attribute certificates

attribute certificate: data structure, digitally signed by an attribute authority, that binds some attribute values with identification information about its holder

certificate: See public key certificate.

certificate identifier: unambiguous identifier of a Certificate

certificate path (chain) validation: process of verifying and confirming that a certificate path (chain) is valid

certificate revocation list: signed list indicating a set of certificates that are no longer considered valid by the certificate issuer

certificate validation: process of verifying and confirming that a certificate is valid

certification authority: authority trusted by one or more users to create and assign certificates

claimed signing time: time of signing claimed by the signer which on its own does not provide independent evidence of the actual signing time

(signature) commitment type: signer-selected indication of the exact implication of a digital signature

(signature) creation constraints: abstract formulation of rules, values, ranges and computation results that are used when creating a digital signature

cryptographic suite: combination of a signature scheme with a padding method and a cryptographic hash function

detached (digital) signature: detached (digital) signature is a type of digital signature that is kept separate from its signed data

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

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

driving application: application that uses a signature creation system to create a signature or a signature validation application in order to validate digital signatures

electronic document: any content stored in electronic form, in particular text or sound, visual or audiovisual recording

evidence: information that can be used to resolve a dispute about various aspects of authenticity of archived data objects

evidence record: unit of data, which can be used to prove the existence of an archived data object or an archived data object group at a certain time

NOTE: See IETF RFC 4998 [i.9] and IETF RFC 6283 [i.10].

proof of existence: evidence that proves that an object existed at a specific date/time, which may be a date/time in the past

public key certificate: public key of an entity, together with some other information, rendered unforgeable by digital signature with the private key of the certification authority which issued it

secure signature creation device: As defined in Directive 1999/93/EC [i.15].

signature attribute: signature property

signature augmentation: process of incorporating to a digital signature information aiming to maintain the validity of that signature over the long term

NOTE: Augmenting signatures is a co-lateral process to the validation of signatures, namely the process by which certain material (e.g. time stamps, validation data and even archival-related material) is incorporated to the signatures for making them more resilient to change or for enlarging their longevity.

signature augmentation policy: set of rules, applicable to a single digital signature or to a set of interrelated digital signatures, that defines the technical and procedural requirements for their upgrade, in order to meet a particular business need, and under which the digital signatures can be determined to be conformant

signature creation application: application within the signature creation system that creates a digital signature, excluding the signature creation device

signature creation data: unique data, such as codes or private cryptographic keys, which are used by the signer to create a digital signature

signature creation device: configured software or hardware used to implement the signature creation data and to create a digital signature

signature creation environment: physical, geographical and computational environment of the signature creation system

signature creation policy: set of rules, applicable to a single digital signature or to a set of interrelated digital signatures, that defines the technical and procedural requirements for their creation, in order to meet a particular business need, and under which the digital signatures can be determined to be conformant

signature creation system: overall system, consisting of the signature creation application and the signature creation device, that creates a digital signature

signature invocation: non-trivial interaction between the signer and the SCA or QSCD/SSCD/SCDev that is necessary to invoke the start of the signing process in the SCA/QSCD/SSCD/ SCDev to generate the Signed Data Object

NOTE: It is the 'Wilful Act' of the signer.

signature policy: signature creation policy, a signature augmentation policy, a signature validation policy or any combination thereof

signature scheme: triplet of three algorithms composed of a signature creation algorithm, a signature verification algorithm and a key generation algorithm

signature validation: process of verifying and confirming that a signature is valid

signature validation application: application that implements signature validation

(signature) constraints: abstract formulation of rules, values, ranges and computation results that a digital signature can be validated against

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

signature validation policy: set of rules, applicable to a single digital signature or to a set of interrelated digital signatures, that defines the technical and procedural requirements for their validation, in order to meet a particular business need, and under which the digital signatures can be determined to be valid

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

signature verification data: data, such as codes or public cryptographic keys, which are used for the purpose of verifying signature

signature verification device: configured software or hardware used to implement the signature-verification data

signer: entity being the creator of a digital signature

time-assertion: time-stamp token or an evidence record

time-stamp token: data object defined in IETF RFC 3161 [3], representing a time-stamp

trust service: electronic service which enhances trust and confidence in electronic transactions

trust service status list: form of a signed list as the basis for presentation of trust service status information

validation: process of verifying and confirming that a certificate or a digital signature is valid

validation data: data that is used to validate a digital signature

verifier: entity that wants to validate or verify a digital signature

3.2 Abbreviations

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

AC	Attribute Certificate
CA	Certification Authority
CRL	Certificate Revocation List
DA	Driving Application
DTBS	Data to be Signed
DTBSF	Data To Be Signed (Formatted)
DTBSR	Data To Be Signed Representation
ERS	Evidence Record Syntax
HTML	HyperText Markup Language
LCP	Lightweight Certificate Policy
LDAP	Lightweight Directory Access Protocol
LTV	Long Term Validation
NCP	Normalized Certificate Policy
OCSP	Online Certificate Status Protocol
ODA	Office Document Architecture
OID	Object Identifier
PKC	Public Key Certificate
PKIX	Public Key Infrastructure X. 509
POE	Proof Of Existence
QCP	Qualified Certificate Policy
QSCD	Qualified electronic Signature/Seal Creation Device
RSA	Rivest, Shamir and Adleman algorithm
SAV	Signature Acceptance Validation
SCA	Signature Creation Application
SCD	Signature Creation Data
SCDev	Signature Creation Device
SCE	Signature Creation Environment
SCS	Signature Creation System
SD	Signer's Document
SDO	Signed Data Object
SDR	Signer's Document Representation
SGML	Standard Generalized Markup Language
SSCD	Secure Signature Creation Device

SVA	Signature Validation Application
TSA	Time-Stamping Authority
TSL	Trust-service Status List
TSP	Trust Service Provider
URI	Uniform Resource Identifier
VCI	Validation Context Initialization
XML	Extensible Mark-up Language
XSL	eXtensible Stylesheet Language

4 Signature creation

4.1 Signature creation model

The objective of signature creation is to generate a signature covering the Signer's Document (SD), the signing certificate or a reference to it, as well as signature attributes supporting the signature and its interpretation and purpose.

The present document uses the functional model of a Signature Creation Environment (SCE) consisting of:

- a signer that wants to create a signature in a document;
- a Driving Application (DA) which represents a user environment (e.g. a business application) that the signer uses to access signing functionality; and
- a Signature Creation System (SCS) which implements the signing functionality.

NOTE: The involvement of a human signer is not always needed; signing may be an automated process implemented in the DA.

Figure 1 illustrates this model. It does not distinguish between hardware or software implementations, and the model does not specify the nature of any inputs/outputs or information transfer paths between the different 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 are implementation issues which are out of scope of the present document.

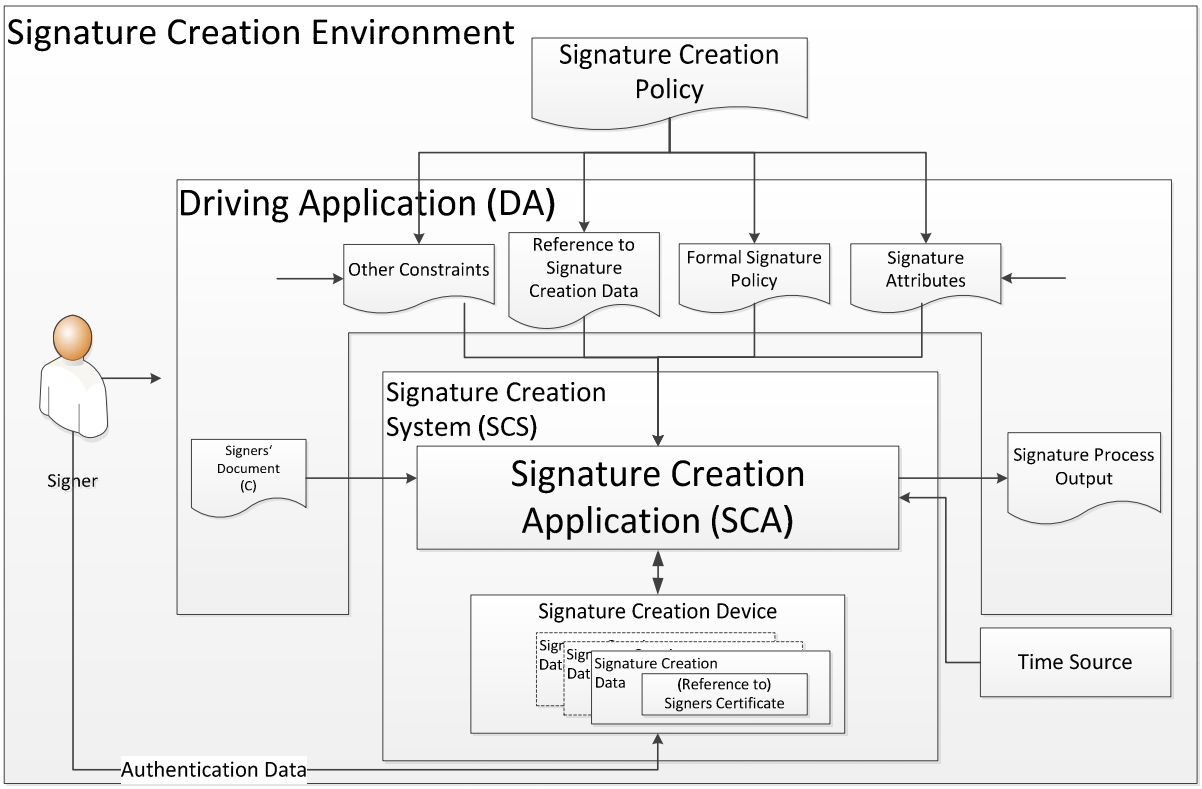

Figure 1: Functional Model of Signature Creation

The Signature Creation System (SCS) contains:

- a Signature Creation Application (SCA); and
- a Signature Creation Device (SCDev).

Clauses 4.2 and 4.3 will specify the details of the signing process, which will consist of the following steps:

- the SCS receives the document to be signed together with other input from the DA;
- composes this into Data To Be Signed (DTBS);
- formats this into Data To Be Signed (Formatted) (DTBSF);
- produces a signature over the DTBSF;
- formats the result into a Signed Data Object (SDO) conforming to the desired signature format (e.g. CAdES [i.2], XAdES [i.4] and PAdES [i.6]); and
- returns the SDO and a status indication to the DA.

In case of an error, the SCS should return additional information allowing the DA or the signer to properly deal with the error.

The signature creation device (SCDev):

- shall hold the signing certificates (or unambiguous references to them);
- holds the corresponding signature creation data;
- shall be able to verify the signer's authentication data; and
- shall create the signature value using the signer's signature creation data.

NOTE: 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.

4.2 Signature creation information model

4.2.1 Introduction

Figure 2 outlines the building blocks for creating a signature and illustrates the data flow for the process of the generation of a signature. Clauses 4.2.2 to 4.2.11 specify information objects used in this process.

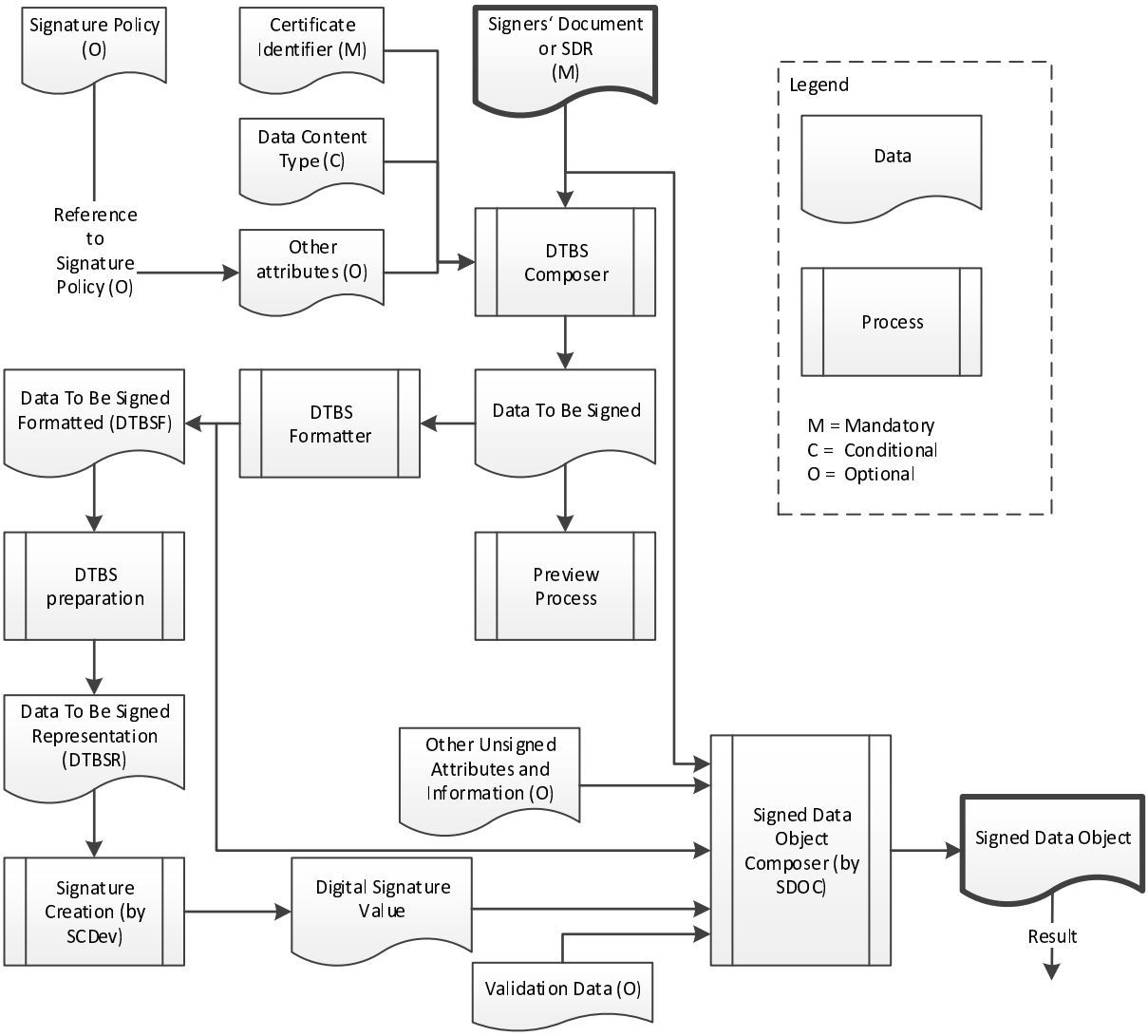

Figure 2: Information Model of Signature Creation

4.2.2 Signature Creation Constraints

The signature creation process shall be controlled by a set of creation constraints. These constraints may be defined:

- using a formal policy specification, e.g. a (machine processable) signature creation policy;
- explicitly in system specific control data: e.g. in conventional configuration-files like property or .ini-files or stored in a registry or database; or
- implicitly by the implementation itself.

Additional constraints may be provided by the DA to the SVA via parameters selected by the application or the user. These constraints influence the creation process and the creation result, irrespective of where these constraints have been defined.

To simplify the specification, these creation constraints are assumed to always be present. Thus, they are omitted as parameters to the different creation blocks described below.

4.2.3 Signer's document (SD)

The Signer's Document (SD) is the document upon which the signature will be generated and to which it will be associated. The SD is selected or composed by the signer or by the DA. In some cases, a Signer's Document Representation (SDR) of the SD can be presented to the signature processes instead of the complete SD.

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

1) It can 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 can be in an unambiguous form (e.g. txt, Postscript, ODA final form, etc.). 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 can be present (e.g. macros, hidden text, active or calculated components, viruses, etc.). These can be invisible to the signer during the preview and verification processes, and the signer can be unaware of their presence. These represent potential ambiguities in the SD.

4) It can be in a form that is not normally presented to the signer or verifier directly, or it can 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.

4.2.4 Signer's document representation (SDR)

The SDR is used in the calculation of the signature as a representation of the SD. The SDR may be provided by the DA to the SCA. Whenever the DA does not provide the SDR, the SCA shall calculate the SDR from the SD by applying the algorithm specified by the signature creation policy in use.

It shall be infeasible to find another SD that is represented by the same SDR.

NOTE: Some signature formats do not incorporate the SD directly into the signature. An SDR typically is built on a cryptographic hash of the SD.

4.2.5 Signature attributes

4.2.5.1 General requirements

Signature attributes shall be pieces of information that support the AdES signature and its interpretation and purpose and which may be covered by the signature together with the SD. The signature attributes shall be either directly provided by the signer or selected through the DA or automatically inserted into the signature by the SCS.

Attributes shall 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 augmenting a signature, by the signature augmentation policy (ETSI TS 119 172-1 [4]) used and can also be format specific.

Clauses 4.2.5.2 to 4.2.5.10 specify signature attributes that are commonly used. Examples of this information and its uses are contained in the ETSI AdES signatures specifications ETSI TS 119 122-1 [i.2], ETSI TS 119 132-1 [i.4], ETSI TS 119 142-1 [i.6].

A signature may contain other signature attributes that are application-specific.

4.2.5.2 Signing certificate identifier

This attribute shall be a signed attribute.

This attribute shall contain one reference to the signing certificate.

NOTE 1: This attribute prevents substitution of the referenced certificate with another one with different semantics but the same public key. 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 some of or all the certificates within the signing certificate path, including one reference to the trust anchor when this is a certificate.

NOTE 2: If so, these references identify a set of certificates that are recommended as the certificate chain used to validate the signing certificate.

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

4.2.5.3 Signature policy identifier

This attribute shall be a signed attribute.

The signature policy identifier attribute may contain a unique identifier for a signature policy. The attribute may also indicate that there is an implied signature policy that the relying party should be aware of.

NOTE 1: See AdES digital signatures specifications for additional requirements.

NOTE 2: This attribute can be present if required by the signing context (e.g. in a specified trading agreement). This identifier indicates to the verifier which is the correct signature policy to be used during the validation process. For instance, a signature policy can be used to clarify the precise role and commitments that the signer intends to assume with respect to the SD.

This attribute shall also contain a digest whose value is computed on the signature policy document, together with an identifier for the algorithm used to calculate that digest, or a transformed version of the signature policy document.

4.2.5.4 Signature policy store

This attribute shall be an unsigned attribute.

This attribute shall hold:

- the signature policy document which is referenced in the signature policy identifier attribute so that it can be used for offline and long-term validation; or
- a URI referencing a local store where this document can be retrieved.

The attribute shall contain an identifier of the syntax used for producing the signature policy document.

4.2.5.5 Data content type

This attribute shall be a signed attribute.

The data content type attribute shall indicate the type of the SD.

NOTE: Additional attributes can specify additional information about the signed document. For instance, when presenting signed data to a human user, having no ambiguity as to the presentation of the signed data object to the relying party can be important. In order for the appropriate representation (e.g. text, sound or video) to be selected by the relying party, information on the content type can be indicated by the signer.

4.2.5.6 Commitment type indication

This attribute shall be a signed attribute.

This attribute shall indicate commitment(s) made by the signer when signing certain documents.

The commitment type indicated shall be expressed in form of either an OID or a URI. It may contain a sequence of qualifiers providing more information about the commitment.

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.

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

4.2.5.7 Counter signatures

This attribute shall be an unsigned attribute.

This attribute shall contain one countersignature of the signature.

NOTE: 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 documents. Each additional signature can sign in turn the latest previously generated signature, or all the previously generated signatures together with the signed document.

4.2.5.8 Claimed signing time

This attribute shall be a signed attribute.

This attribute shall contain the time at which the signer claims to having performed the signing process.

NOTE: As is the case with paper based signatures, the time of the signature can only be a claimed one. Methods like time stamps will need to be used in case the signature policies require more than claims for the signature time.

4.2.5.9 Claimed signer location

This attribute shall be a signed attribute.

This attribute shall specify an address associated with the signer at a particular geographical (e.g. city) location where the signer claims to having produced the signature.

NOTE: In some transactions the place where the signer was at the time of signature creation is needed.

4.2.5.10 Signer's attributes

This attribute shall be a signed attribute.

This attribute shall contain attributes or signed assertions that the signer claims or proves to be in possession of when the signature was generated. This shall be done using:

- a signer's claimed attribute;
- an attribute certificate issued by an attribute authority; or
- signed assertions issued by a service provider.

NOTE: 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 can only be valid if signed by a user in a particular role, e.g. a sales director. In many cases it is not that important to know who the sales director really is, but being sure that the signer is empowered by his company to be the sales director can be fundamental.

4.2.6 Data to be signed (DTBS)

The data to be signed shall consist of the information objects that are to be covered by the signature. These are:

- the SD or the SDR; and
- the signature attributes selected to be signed together with the SD.

NOTE: A signature is typically made over the SDR and some attributes. When there are no attributes, some signature formats allow to use the SDR directly when creating the signature.

4.2.7 Data to be signed (formatted) (DTBSF)

The DTBSF shall be created from the DTBS objects by formatting them and placing them in the correct sequence for the signing process.

NOTE: This step can include processes like canonicalization/normalization or XSL transformation. The result of this step is the information object that is covered by the signature as the result of the signing processes and which is included in the signature calculation and verification. A signature is typically made over the SDR and some attributes. When there are no attributes, some signature formats allow to use the SDR directly when creating the signature.

4.2.8 Data to be signed representation (DTBSR)

The DTBS preparation component shall take the DTBSF and hash it according to the hash algorithm specified in the cryptographic suite. The result of this process is the DTBSR, which will then be used to create the signature.

NOTE: In order for produced hash to be representative of the DTBSF, the hashing function has the property that it is computationally infeasible to find collisions for the expected signature lifetime. Should the hash function become weak in the future, additional security measures, such as applying time-stamp tokens, can be taken.

4.2.9 Signature

The SCDev shall take the DTBSR and apply the signature algorithm specified in the cryptographic suite. The result of this process shall be the signature value.

4.2.10 Signed data object (SDO)

The Signed Data Object Composer shall produce a SDO by taking the signature value calculated and formatting it according to the SDO Type. The SDO shall contain:

- the signature value; and
- the signed attributes.

The SDO may additionally contain the following:

- the SD or SDR; and/or
- additional supportive unsigned attributes.

4.2.11 Validation data

Some classes of AdES signatures incorporate additional data needed for validation. This additional data is called validation data, is the result of a signature augmentation process and shall include:

- Public Key Certificates (PKCs);
- revocation status information for each PKC (Certificate Revocation Lists (CRLs) or certificate status information (OCSP)); and
- time-assertions applied to the signature.

Validation data may also include other additional data necessary or useful for validation like Attributes Certificates (ACs) and revocation status information for ACs.

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

4.3 Signature Classes and Creation Processes

4.3.1 Introduction

Figure 3 illustrates the structure of a signature common to all classes of signatures that will be defined in this clause. It consists of the signer's document and signed attributes, both of which are input to the calculation of the signature value, the signature value itself as well as any unsigned attributes included into the signature.

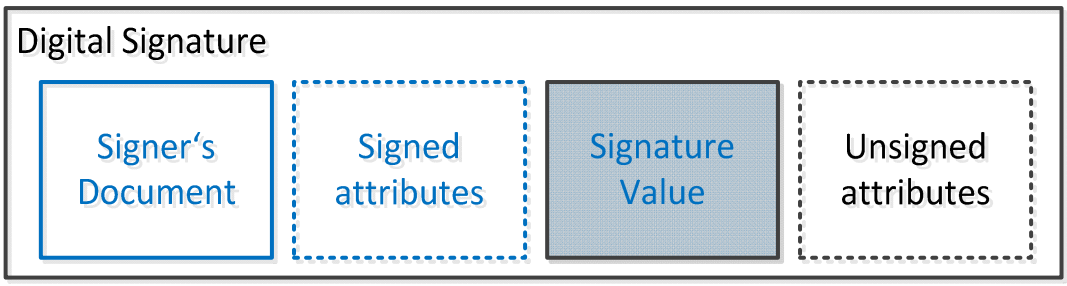

Figure 3: Digital Signature

Figure 4 illustrates the life cycle of a signature. Most signatures created will only encounter some of the steps in the life cycle. The steps in the life cycle are defined here as classes of signatures that have common properties as specified below. The process of creating an instance of a signature class based on a signature of another class following that lifecycle is also called **Signature Augmentation** and is governed by a signature augmentation policy.

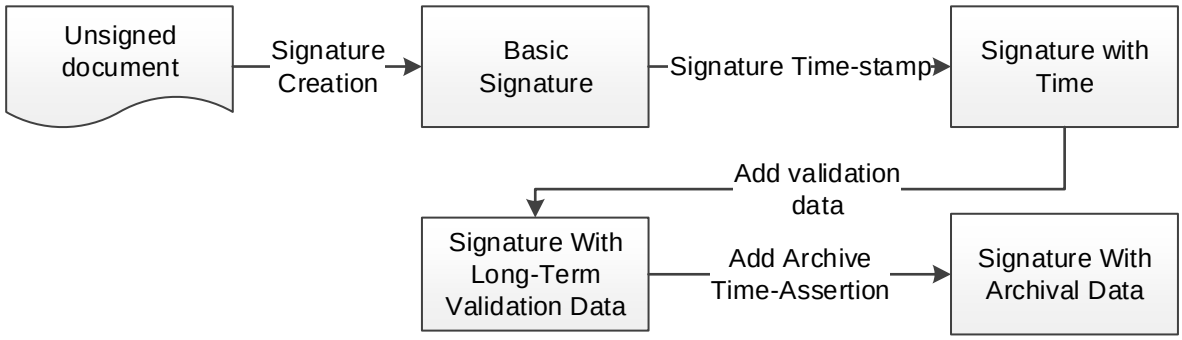

Figure 4: Signature Lifecycle

Each of the signature classes below corresponds to a meaningful combination of attributes added to a signature aiming at improving the ability to validate a signature in the future, when the corresponding certificate or any other material needed for successful validation may have expired, been revoked, or used algorithms are no longer strong enough to be trustworthy.

A **Basic Signature** is a signature that can be validated as long as the corresponding certificates are neither revoked nor expired.

A **Signature with Time** is a signature that proves that the signature already existed at a given point in time.

NOTE 1: It can be used to validate a signature when a certificate has been revoked after the signature has been created.

A **Signature With Long-Term Validation Data** is a signature that provides the long term availability of the validation material by incorporating all the material or references to material required for validating the signature.

A **Signature With Archival Data** ensures that the validation material provided with the signature is kept preserved for even longer term. This level aims to tackle the long term availability and integrity of the validation material.

NOTE 2: Signatures can then still be validated when certificates expire or become revoked, and also when the security of applied algorithms becomes questionable or used key sizes are no longer state of the art.

4.3.2 Creation of Basic Signatures

4.3.2.1 Description

Figure 5 shows the steps involved in creation of a Basic Signature. Clauses 4.3.2.4.1 to 4.3.2.4.7

specify these steps.

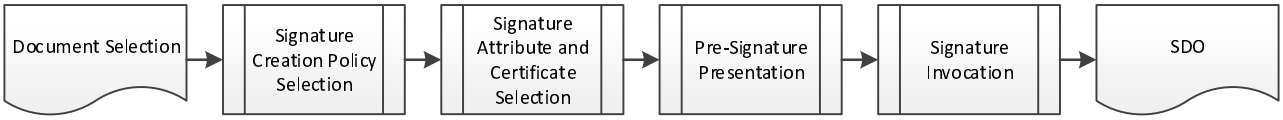

Figure 5: Basic Signature Creation

4.3.2.2 Inputs

Table 1: Inputs to the Basic Signature Creation process

Input	Requirement
Signer's Document or Signer's Document Representation	Mandatory
Signing Certificate	Mandatory
Other Signature Attributes	Optional
Signer's Authentication Data	Mandatory
Signature Creation Policy	Optional

NOTE: A signature can also contain the time the signature has been created. It is assumed that the current time is accurately available to the SCA. It is not listed as an input to avoid giving the impression that this time value can be selected at will.

4.3.2.3 Outputs

The output of the Basic Signature creation process is an SDO that shall contain

- the signature value;
- a reference to or a copy of the signing certificate as a signed attribute; and
- any optional signed or unsigned attributes (e.g. a signature policy identifier (see clause 4.2.5.3).

NOTE 1: A Basic Signature is designed to prevent simple substitution and reissuing attacks and to specify the certificate to be used for verifying the signature.

NOTE 2: Additional mandatory attributes can be format specifically defined.

Figure 6 illustrates a Basic Signature.

Figure 6: Basic Signature

4.3.2.4 Processing

4.3.2.4.1 Selection of documents to sign

The Driving Application shall select one or more documents to be signed either automatically or explicitly by the signer through a user interface.

The selection process may specify that only certain parts of a document are to be signed.

NOTE: Legal requirements can mandate explicit signer involvement in selection of document to sign.

When a document is selected for signing, any existing signature on or attached to the document should be validated. If the signature is validated, a warning shall be provided in case validation of an existing signature yields a *TOTAL-FAILED* or *INDETERMINATE* result.

4.3.2.4.2 Signature attribute and parameters selection

The signing certificate identifier attribute (see clause 4.2.5.2) shall be included in the DTBS whenever required by the format and the contents of the signature. Other attributes may be included in the DTBS or as unsigned attributes in the resulting SDO.

NOTE 1: in XAdES signatures, it is possible to sign the signing certificate present within the ds:Keyinfo element. In this case, the signing certificate identifier attribute is not required.

The signing process shall be guided by additional parameters that at least shall determine the format of the SDO (e.g. CAdES, XAdES, PAdES) and the class (e.g. Basic Signature, Signature with Time, etc.) of the signature to create, and when necessary whether a detached, enveloped or enveloping signature shall be created.

Certificate and other attributes and parameters shall be selected by one or a combination of:

- The DA, conveying the attributes as parameters over the interface to the SCA.
- The user through a user interface offered by the DA or the SCS.
- Local configuration in the SCS. Or
- Other means, such as importing a Signature Policy to the DA or the SCS.

NOTE 2: The signing certificate, possibly the complete certificate path, can most often be obtained from the user's SCDev.

4.3.2.4.3 Pre-signature presentation

When specific legal or functional requirements require that a document to sign is presented to the user, the DA shall present the document to sign. In other cases, the DA should enable the user to inspect the document to sign.

NOTE 1: Presentation of document to sign is not always convenient. One example is bulk signing of many documents in one operation (e.g. invoices). Another example is signing of medical prescriptions where the user dialogue presents the medication information but not necessarily the final prescription documents (several prescriptions for one patient can be signed following one user consent).

The presented document shall be equal in content to the document that is signed.

NOTE 2: "Equal in content" is used instead of "the same", as the representation that is signed can be another format than the one displayed (e.g. XML).

The DA should enable the user to inspect attributes selected for the signing process.

The DA may rely upon the SCS for whole or parts of the presentation of document to sign and attributes.

When a document to sign is presented, the validation result of any existing signature on or attached to the document should be presented to the user as per clause 4.3.2.4.1.

The SCA may allow the user to inspect a document to sign and/or attributes selected for the signing process through a separate user interface, outside of the DA environment.

4.3.2.4.4 Signature invocation

When specific legal or functional requirements require user consent prior to signature invocation, the DA shall:

a) follow the procedure as per clause 4.3.2.4.3;

b) inform the user of the implications of signing; and

c) get consent from the user.

The DA may rely upon the SCS for whole or parts of the user dialogue.

Once user consent has been received, the DA shall invoke the signature to the SCS/SCA.

4.3.2.4.5 Signing

The SCDev shall perform the signing operation.

NOTE: See clause 4.3.2.4.6 for additional requirements on user authentication for the activation of the signature creation data.

Before invoking use of the signature creation data, the SCS (SCA or SCDev) should check that the signing certificate is valid (cryptographically correct, within its validity period and not revoked).

When the SCDev returns the signature, the SCA should verify the signature using the public key from the user's certificate.

4.3.2.4.6 Signer authentication

The use of the signer's signature creation data (the private key) in the SCDev may require the user to enter specific signer authentication data.

The user may input the authentication data in a user interface to the DA, to the SCA, or to the SCDev.

More than one authentication mechanisms may be used to provide sufficient authentication assurance.

A signer authentication mechanism may be of a form that prevents impersonation attacks even from the DA or the SCA and their environment.

NOTE 1: The nature of authentication mechanisms and the related signer authentication data are determined by the SCDev used. Standards exist for different interfaces, SCDev types, and authentication mechanisms.

NOTE 2: In some cases, use of signer authentication data will be mandatory and further requirements on the nature of the authentication mechanisms and interfaces can be imposed.

4.3.2.4.7 SDO composition

Upon return of the signature from the SCDev, the SCA shall compose the SDO according to the required format. Further attributes may be included in the SDO.

A status indication shall be returned to the DA.

The status indication shall have one of two values:

- *OK*: The signature has been successfully created; in this case, the SDO shall also be returned to the DA;
- *FAILED*: The SCS was unable to create a signature.

In case of an error, the SCS should return additional information allowing the DA or the signer to properly deal with the error.

4.3.3 Creation of a Signature with Time

4.3.3.1 Description

A Signature with Time is a signature that proves that the signature already existed at a given point in time. The time is provided by a time-stamp token on the signature as an unsigned property added to the Basic Signature as a result of the signature augmentation.

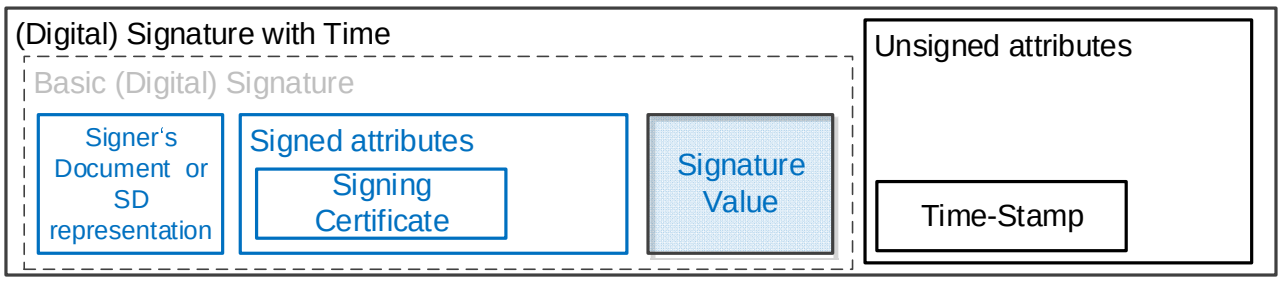

Figure 7: Signature with Time

NOTE 1: 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 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.

NOTE 2: Time-stamp token provides the initial steps towards providing long term validity. The time-stamp tokens need to be created before a certificate has been revoked or expired. If this cannot be achieved, validation of the created signature can fail.

NOTE 3: The Signature with Time provides independent evidence of the existence of the signature prior to the time-stamp token indication. To reduce the risk of repudiating signature creation, the time-stamp token ideally is as close as possible to the time the signature was created. The signer or a TSP could provide the Signature with Time. If the signer did not provide it or the TSA the signer used is not trusted by the verifier, the verifier can create a Signature with Time on first receipt of a signature.

4.3.3.2 Inputs

Table 2: Inputs to the creation process for Signatures with Time

Input	Requirement
Basic Signature	Mandatory
Signature Augmentation Policy	Optional

4.3.3.3 Outputs

The process for creating a **Signature With Time** shall return the signature provided with an added unsigned attribute containing a time-stamp token on the signature.

4.3.3.4 Process

The signature augmentation process shall:

1) Request one or more time-stamp tokens from appropriate TSAs as defined in the signature policy or local configuration. The time-stamp token shall cover the signature value.

2) Produce a signature attribute encapsulating the time-stamp token(s) produced in step 1. And

3) Add the signature attribute of step 2 as an unsigned attribute to the SDO.

NOTE: When the validation of a signature fails, adding a time-stamp will not always help preserving the signature. The SVA can optionally validate the signature before requesting a time-stamp and in case of *TOTAL-FAILED* abort the process.

4.3.4 Creation of Signatures With Long-Term Validation Data

4.3.4.1 Description

As long as a validation algorithm can assess the validity of a Signature With Time, it can be augmented to a Signature With Long-Term Validation Data by adding unsigned attributes. This augmentation can be done either by the SCA, or by a third party or by a verifier using a SVA.

NOTE 1: A signature validation algorithm can assess the validity of a Signature With Time only as long as the validation data required to validate the signature is still on-line available to the verifiers. In case it is unsure that the validation data required to validate the signature will still be on-line available to the verifiers or that some verifiers cannot access that data, then it is necessary to capture that data inside the signature.

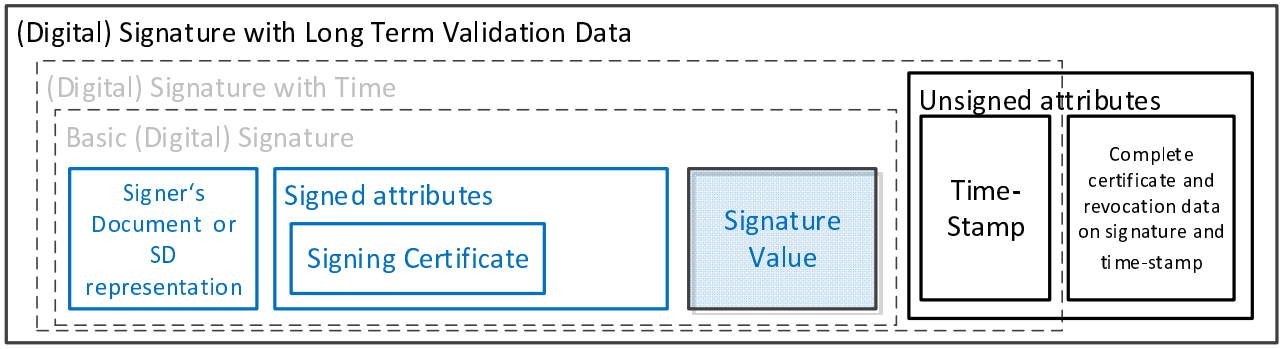

Figure 8: Signature with long term validation data

NOTE 2: A signature with long term validation data includes the validation data that is necessary to verify the signature beyond the end of the validity of the signing certificate, in particular to ascertain the revocation status of all end-entity certificates (signing certificate, time-stamping units certificates, attribute certificates ...) contained in the signature.

There can be more elements than necessary and can also be fewer elements than necessary if it is expected that recipients have an alternate means of obtaining the missing elements.

4.3.4.2 Inputs

Table 3: Inputs to the creation process for Signatures with Long Term Validation Data

Input	Requirement
Signature With Time	Mandatory
Signature Augmentation Policy	Optional

4.3.4.3 Outputs

If the signature provided can be validated, the process for creating a Signature with Long Term Validation Data shall return a success indication together with the created signature with long term validation data.

Otherwise, the process shall return a failure indication together with all information about the problem as provided by the validation process.

4.3.4.4 Process

When adding an attribute containing long-term-validation data, the signature augmentation process shall:

1) validate the signature with time in its current state, using the validation defined in clause 5.5;

2) if the validation process returns *TOTAL-FAILED*, return this indication together with all information about the problem as provided by the validation process;

3) if the validation process returns *TOTAL-PASSED*, add to the signature all material or/and references to it that has been used during validation and that is not already present in the signature and shall return the resulting signature with a success indication; and

4) if the validation process returns *INDETERMINATE*, add to the signature all material or/and references to it that has been used during validation and that is not already present in the signature and shall return the signature together with the validation status information.

4.3.5 Creation of Signatures with Archival Data

4.3.5.1 Description

Before algorithms, keys, and other cryptographic data used at the time a signature was built become weak and the cryptographic functions become vulnerable, or the certificates supporting previous time-assertions expire or are revoked, the signer's document, the signature as well as any attributes contained in a signature with long term validation data should be protected by applying one or more time-assertions. Time-assertions bind data to a particular time establishing evidence that the latter data existed at that time. Such additional time-assertions are called archive validation data. The creation of time-assertions should be repeated in time before the protection provided by a previous time-assertion becomes weak and should make use of stronger algorithms or longer key lengths than have been used in the original signatures or previous time-assertions.

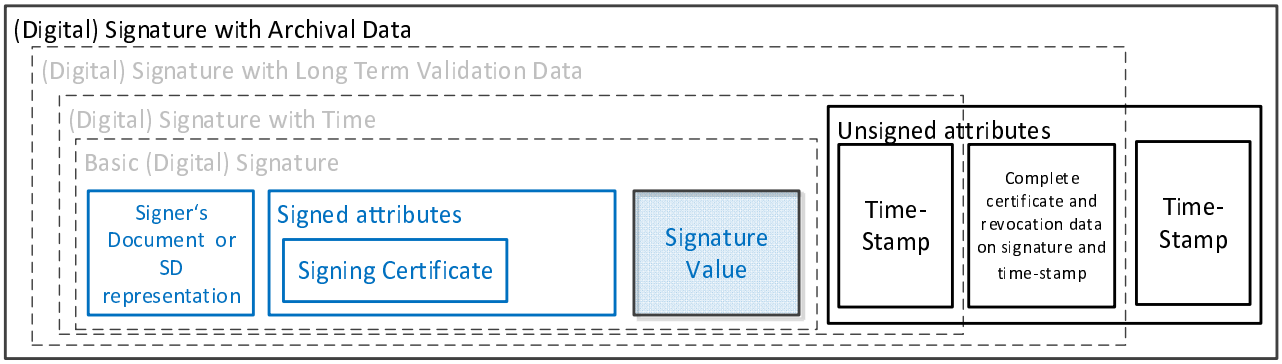

Figure 9: Signature with Archival Data

If the process is repeated, several instances of archive time-stamps may occur with a signature. Figure 10 shows an example of a signature where two time-assertions have been applied.

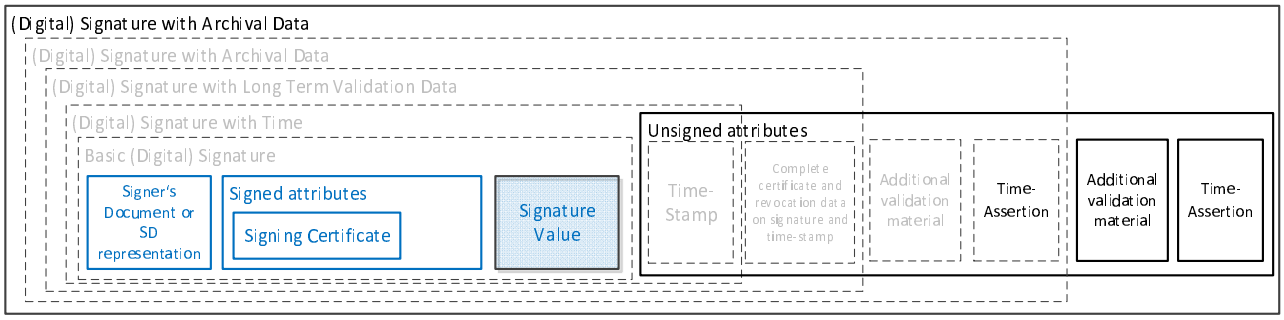

Figure 10: Signature with Archival Data after repetition

4.3.5.2 Inputs

Table 4: Inputs to the creation process for Signatures with Archival Data

Input	Requirement
Signature with Long Term Validation Data or Signature with Archival Data	Mandatory
Signature Augmentation Policy	Optional

4.3.5.3 Outputs

The process for creating a **Signature With Archival Data** shall, if successful, return the signature provided as input which has been augmented by an added unsigned attribute containing an archive time-assertion, e.g. a time-stamp token or an evidence record on the signature. Also, additional validation material may have been included as unsigned attributes within the signature.

If the process has not been executed successfully, an error indication shall be returned together with all information available explaining the error.

4.3.5.4 Process

The signature augmentation process shall:

1) Validate the signature in its current state, using the validation defined in clause 5.5.

2) If the validation process returns TOTAL-FAILED, return this indication together with all information about the problem as provided by the validation process.

3) Add any validation material required for validating the signature that is not already present in the signature. This shall include any validation data of previously added time-assertions.

4) Request one or more time-assertions from appropriate TSAs as defined in the signature policy or local configuration. The time-assertion shall cover the signer's document as well as all data objects contained in the signature.

5) Produce signature attribute(s) encapsulating the time-assertion(s) produced in step 4. And

6) Add the signature attribute(s) as unsigned attribute(s) to the signature.

5 Signature validation

5.1 Signature validation model

5.1.1 General Requirements

This clause defines the conceptual model shown in Figure 11 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 an AdES signature and other input from the driving application (DA).

The SVA shall validate the AdES signature against a signature validation policy, consisting of a set of validation constraints, and shall output a status indication and validation report providing the details of the technical validation of each of the applicable constraints, which can be relevant for the DA in interpreting the results.

The status of each single validation building block shall be:

PASSED: when all checks that the signature validation policy prescribed have been passed for the particular validation building block.

INDETERMINATE: when the available information is insufficient to full process all checks that the signature validation policy prescribed have been passed for the particular validation building block.

FAILED: if not *PASSED* or *INDETERMINATE.*

The status on the full validation in the context of a particular signature validation policy shall be:

TOTAL-PASSED: when all checks that the signature validation policy prescribed have been passed.

TOTAL-FAILED: all checks failed, or the cryptographic checks of the signature failed, (including checks of hashes of individual data objects that have been signed indirectly) or it is proved that the generation of the signature was after the revocation of the signing certificate; or

INDETERMINATE: the available information is insufficient to ascertain the signature to be *TOTAL-PASSED* or *TOTAL-FAILED.*

The main status indication can be accompanied by additional information. Detailed requirements are specified in clause 5.1.3.

The output of the SVA is meant to be processed by the DA (e.g. to be presented to the verifier).

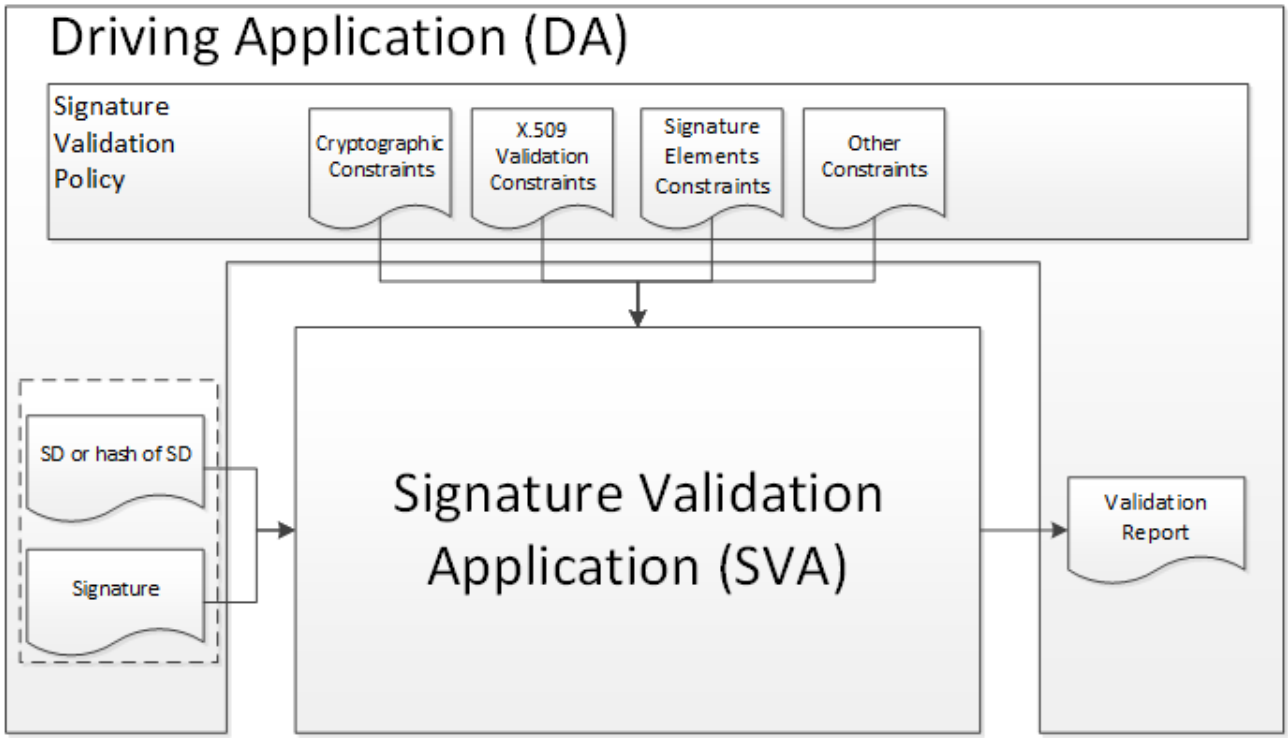

Figure 11: Conceptual Model of Signature Validation

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. However:

- If SVA returns *TOTAL-PASSED* for a certain signature, DA should consider the signature as a technically valid signature according to the validation constraints.

NOTE 1: This does not necessarily mean that the signature is useful for a particular purpose.

- If SVA returns *TOTAL-FAILED* or *INDETERMINATE*, the DA should not consider the signature as technically valid. In case of *INDETERMINATE*, the DA may retry validation based on additional information or at a later point of time.

NOTE 2: This assumes that all validation constraints were available to the SVA. There can be cases, where this cannot be done. For example, if the validation constraints state that the claimed signing time of a signature is assumed to be the actual signing time even if there are no proofs of existence for that fact, and the SVA is unable to take this consideration into the decision process, the SVA will return *INDETERMINATE* with an indication for the reason. This will allow the DA to still accept the signature as valid according to the policy in place.

The present document presents the validation process in the form of algorithms which should be implemented by a signature validation application.

The implementations shall produce results equivalent to the results produced by the algorithms given the same set of input information.

NOTE 3: 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.

5.1.2 Selecting validation processes

The clauses below offer several validation processes. Depending on the classes of signatures a SVA is intended to be able to validate, an appropriate validation process needs to be selected:

- When supporting only validation of Basic Signatures, the SVA shall support the **Validation Process for Basic Signatures** (clause 5.3). This process may be selected for signatures where:
  - the time of validation lies within the validity period of the signing certificate and the signing certificate has not been revoked; or
  - the time of validation lies beyond the validity period of the signing certificate when the certification authority provides revocation information for expired certificates.

NOTE: Validation of signatures where certificates are involved that are expired at validation time, but not revoked, depends on the signature validation policy in use. A policy may e.g. well allow using the Validation Process for Basic Signatures for validating a signature that has been created two days ago, and the involved certificate expired yesterday.

This may be done irrespective of the class of signature presented: Basic Signatures, Signatures with Time, Signatures with Long Term Validation Data and Signatures with Archival Data. Any additional material present in attributes may be ignored.

Certificate and revocation information collected during that validation may be used to create a Signature with Long Term Validation Data. Signature- and other time-stamps should only be applied after a successful validation.

- When supporting validation for Basic Signatures and Signatures with Time, the SVA shall support the **Validation Process for Basic Signatures** (clause 5.3) and the **Validation Process for Signatures with Time and Signatures with Long-Term Validation Data** (see clause 5.5).

- When supporting validation for Basic Signatures, Signatures with Time and Signatures with Long Term Validation Data, the SVA shall support the **Validation Process for Basic Signatures** (clause 5.3), the **Validation Process for Signatures with Time and Signatures with Long-Term Validation Data** (see clause 5.5). The SVA shall also be able to use the validation data stored within the signature for validation.

- When supporting validation for Basic Signatures, Signatures with Time, Signatures with Long Term Validation Data and Signatures with Archival Data, the SVA shall support the **Validation Process for Basic Signatures** (clause 5.3), the **Validation Process for Signatures with Time and Signatures with Long-Term Validation Data** (see clause 5.5) and the **Validation process Signatures with Archival Data** (see clause 5.6).

When validating an instance of a signature, the SVA should select the supported process best suited for that signature. This means:

- When validating a Basic Signature, the SVA should select the Validation Process for Basic Signatures.
- When validating a Signature with time, the SVA should select the Validation Process for Signatures with Time and Signatures with Long-Term Validation Data, if supported Otherwise, it should select the Validation Process for Basic Signatures.
- When validating a Signature with Long-Term Validation Data, the SVA should select the Validation Process for Signatures with Time and Signatures with Long-Term Validation Data, if supported. Otherwise, it should select the Validation Process for Basic Signatures.
- When validating a Signature with Archival Data, the SVA should select the Validation process for Signatures with Archival Data, if supported. Otherwise, the SVA should select the Validation Process for Signatures with Time, if supported, else the Validation Process for Basic Signatures.
- Whenever the DA specifies the process to be used, the SVA shall select that process.

Whenever the SVA has no indication which class the signature to be validated belongs to, the SVA should select the process to use as follows:

- If the SVA supports the Validation process for Signatures with Archival Data, it should select that process.
- Otherwise: If the SVA supports the Validation Process for Signatures with Time and Signatures with Long-Term Validation Data, it should select that process.
- Otherwise, the SVA shall select Validation Process for Basic Signatures.

5.1.3 Status indication of the signature validation process and signature validation report

A SVA shall provide a comprehensive report of the validation, 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.

This clause specifies minimum requirements for the content of such a report.

The DA shall, when a human user is involved, be able to present the report in a way meaningful to the user.

In all cases, the signature validation process shall output:

- a status indication of the results of the signature validation process. Table 5 lists the possible values of the main status indication and their semantics;
- an indication of the policy or an indication of the set of constraints against which the signature has been validated;
- the date and time for which the validation status was determined; and
- additional validation report data as specified in Table 5 and Table 6;

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 Signatures with Time, Signatures with Long-Term Validation Data or Signatures with Archival Data.

Indications returned by SVAs shall conform to the following rules:

- When the validation process selected as in clause 5.1.2 returns *PASSED,* the overall result of the validation shall be *TOTAL-PASSED.*
- When the validation process selected as in clause 5.1.2 returns *FAILED*, the overall result of the validation shall be *TOTAL-FAILED.*
- When the validation process selected as in clause 5.1.2 returns *INDETERMINATE,* the overall result of the validation shall be *INDETERMINATE*.
- **When the result is *TOTAL-PASSED* or *TOTAL-FAILED*:**

  a) Any execution of a SVA with the same inputs shall return *TOTAL-PASSED* or *TOTAL-FAILED,* respectively.

  b) Any execution of a SVA with the same inputs and additional validation data (e.g. more certificates) shall return the same result as it has returned in *a)* (i.e. *TOTAL-PASSED* or *TOTAL-FAILED*).

- **When the result is INDETERMINATE:**

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

  b) Any execution of a SVA with the same inputs and additional validation data shall return *TOTAL-PASSED*, *TOTAL-FAILED* or INDETERMINATE.

In the case of an *INDETERMINATE* validation result, an SVA shall provide secondary status information able to identify the first error encountered and may provide additional status information for further constraints that have been checked.

NOTE 2: The date/time at which the SVA is executed is an implicit input to the validation process. Running the SVA at a later point in time can give different results in case additional data becomes available (e.g. new certificate status information).

NOTE 3: The term "same inputs" includes the signature validation policy or set of validation constraints to be used. Different validation constraints will in general result in different validation results. Also, if the SVA fetches validation information from e.g. a CA, this is considered as input to the validation.

Table 5: Status indications of the signature validation process

Status indication	Semantics	Associated Validation report data
TOTAL-PASSED	The signature validation process results into *TOTAL-PASSED* based on the following considerations: • the cryptographic checks of the signature succeeded (including checks of hashes of individual data objects that have been signed indirectly); • any constraints applicable to the signer's identity certification have been positively validated (i.e. the signing 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 validated certificate chain, including the signing certificate, used in the validation process. In addition, the validation process may provide the result of the validation for each of the validation constraints. The validation process should provide the DA access to the signed attributes present in the signature, the identity of the signer and the signing certificate chain.
TOTAL-FAILED	The signature validation process results into *TOTAL-FAILED* because the cryptographic checks of the signature failed (including checks of hashes of individual data objects that have been signed indirectly) or it has been proven that the generation of the signature took place after the revocation of the signing certificate.	The validation process shall output additional information to explain the *TOTAL-FAILED* indication for each of the validation constraints that have been taken into account and for which a negative result occurred.

Status indication	Semantics	Associated Validation report data
INDETERMINATE	The available information is insufficient to ascertain the signature to be *TOTAL-PASSED* or *TOTAL-FAILED*.	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 those validation constraints that have been taken into account and for which an indeterminate result occurred.

The validation report data associated to the *TOTAL-FAILED* and *INDETERMINATE* indications status resulting from the validation of an AdES signature should be structured as in Table 6 by listing the main and sub codes to be returned by the validation process.

Table 6: Validation Report Structure

Main indication	Sub indication	Semantics	Associated Validation report data
TOTAL-FAILED	*HASH_FAILURE*	The signature validation process results into *TOTAL-FAILED* 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 or OID) uniquely identifying the element within the signed data object (such as the signature attributes, or the SD) that caused the failure.
	SIG_CRYPTO_FAILURE	The signature validation process results into *TOTAL-FAILED* because the signature value in the signature could not be verified using the signer's public key in the signing certificate.	The validation process shall output: • The signing certificate used in the validation process.
	REVOKED	The signature validation process results into *TOTAL-FAILED* because: • the signing certificate has been revoked; and • there is PoE available 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, if available, the reason of revocation of the signing certificate.
INDETERMINATE	*SIG_CONSTRAINTS_ FAILURE*	The signature validation process results into *INDETERMINATE* because one or more attributes 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 validation process results into *INDETERMINATE* 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.
	CERTIFICATE_CHAIN_ GENERAL_FAILURE	The signature validation process results into *INDETERMINATE* because he set of certificates available for chain validation produced an error for an unspecified reason.	The process shall output: • Additional information regarding the reason.

Main indication	Sub indication	Semantics	Associated Validation report data
	CRYPTO_CONSTRAINTS_FAILURE	The signature validation process results into *INDETERMINATE* because at least one of the algorithms that have been used in material (e.g. the signature value, a certificate...) involved in validating the signature, or the size of a key used with such an algorithm, is below the required cryptographic security level, and: • this material was produced after the time up to which this algorithm/key was considered secure (if such a time is known); and • the material is not protected by a sufficiently strong time-stamp applied before the time up to which the algorithm/key was considered secure (if such a time is known).	The process shall output: • Identification of the material (signature, certificate) that is produced using an algorithm or key size below the required cryptographic security level. • If known, the time up to which the algorithm or key size were consider secure.
	EXPIRED	The signature validation process results into *INDETERMINATE* because the signing time lies after the expiration date (notAfter) of the signing certificate.	The process shall output: • The validated certificate chain.
	NOT_YET_VALID	The signature validation process results into *INDETERMINATE* because the signing time lies before the issuance date (notBefore) of the signing certificate.	
	FORMAT_FAILURE	The signature validation process results into INDETERMINATE because the signature is not conformant to one of the base standards (e.g. ETSI TS 119 122 part 1 [i.2] and part 2 [i.3], ETSI TS 119 132 part 1 [i.4] and part 2 [i.5], ETSI TS 119 142 part 1 [i.6] and part 2 [i.7], IETF RFC 3852 [i.8], XML DSig [i.11]).	
	POLICY_PROCESSING_ERROR	The signature validation process results into *INDETERMINATE* because a given formal policy file could not be processed for any reason (e.g. not accessible, not parseable, digest mismatch, etc.)	The validation process shall provide additional information on the problem.
	SIGNATURE_POLICY_NOT_AVAILABLE	The electronic document containing the details of the policy is not available-	
	TIMESTAMP_ORDER_FAILURE	The signature validation process results into *INDETERMINATE* because 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.
	NO_SIGNING_CERTIFICATE_FOUND	The signature validation process results into *INDETERMINATE* because the signing certificate cannot be identified.	
	NO_CERTIFICATE_CHAIN_FOUND	The signature validation process results into *INDETERMINATE* because no certificate chain has been found for the identified signing certificate.	

Main indication	Sub indication	Semantics	Associated Validation report data
	REVOKED_NO_POE	The signature validation process results into *INDETERMINATE* because the signing certificate was 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 signing certificate.
	REVOKED_CA_NO_POE	The signature validation process results into *INDETERMINATE* because at least one certificate chain was found but an intermediate CA certificate is 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 signature validation process results into *INDETERMINATE* because the signing 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 signing certificate.	
	CRYPTO_CONSTRAINTS_FAILURE_NO_POE	The signature validation process results into *INDETERMINATE* because at least one of the algorithms that have been used in material (e.g. the signature value, a certificate...) involved in validating the signature, or the size of a key used with such an algorithm, is below the required cryptographic security level, and there is no proof that this material was produced before the time up to which this algorithm/key was considered secure.	The process shall output: • Identification of the material (signature, certificate) that is produced using an algorithm or key size below the required cryptographic security level. • If known, the time up to which the algorithm or key size were consider secure.
	NO_POE	The signature validation process results into *INDETERMINATE* because 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 shall identify at least the signed objects for which the POEs are missing. The validation process should provide additional information on the problem.

Main indication	Sub indication	Semantics	Associated Validation report data
	TRY_LATER	The signature validation process results into *INDETERMINATE* because 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.
	SIGNED_DATA_NOT_ FOUND	The signature validation process results into *INDETERMINATE* because signed data cannot be obtained.	The process should output when available: The identifier (s) (e.g. an URI) of the signed data that caused the failure.
	GENERIC	The signature validation process results into *INDETERMINATE* because of any other reason.	The validation process shall output: • Additional information why the validation status has been declared Indeterminate.

5.1.4 Validation constraints

5.1.4.1 General Requirements

The validation process shall be controlled by a set of validation constraints. These constraints may be defined:

- using a formal policy specification which should be as specified in ETSI TS 119 172-1 [4] or machine processable equivalents;
- explicitly in system specific control data: e.g. in conventional configuration-files like property or .ini-files or stored in a registry or database; or
- implicitly by the implementation itself.

Any validation constraints not implied by the implementation may 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; or
- a local source from the verifier (e.g. configuration file, (machine processable) signature validation policy).

Additional constraints may be provided by the DA to the SVA via parameters selected by the application or the user. These constraints influence the validation process and the validation result, irrespective of where these constraints have been defined.

NOTE 1: 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 IETF RFC 5280 [1]. Details on how to check that the signature matches such constraints are not given in the present document.

If the validation algorithm prescribes a certain check and the set of constraints state that such a check is not required (e.g. revocation checking), an SVA may skip that step and continue as if the check has succeeded. In such cases, the SVA should return, in its final report to the DA, the list of checks that were disabled due to the policy.

NOTE 2: The verifier can select a signature validation policy that contains additional constraints which are not mentioned in the present document. It is not foreseeable, which constraints a DA will impose on the SVA. It is assumed that an implementation handles all constraints properly.

EXAMPLE 1: A validation policy can contain the constraint that the revocation freshness check (5.2.5) is to be skipped. Such cases are not covered by the basic signature validation algorithm in clause 5.3.4since this would make the description less readable.

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

The set of validation constraints used for validation may force the SVA not to check a constraint that, when checked, would, according to the present document, lead to an *INDETERMINATE* result. The SVA shall report such cases in the validation report.

EXAMPLE 2: if validation constraints force the SVA to ignore revocation status of intermediate certificates, the SVA will return *TOTAL-PASSED*, even if it would be expected to return *INDETERMINATE*. Such overruling by the policy is possible for all decisions made by the present document and cannot be mentioned in all places they can appear.

The set of validation constraints used for validation shall not force the SVA not to check a constraint that, when checked, would, according to the present document, lead to an *TOTAL-FAILED* result.

The following constraints shall be supported:

- chain constraints, as defined in clause 5.1.4.2;
- cryptographic constraints as defined in clause 5.1.4.3;
- signature elements constraints as defined in clause 5.1.4.4.

Where other constraints are implemented, their meaning shall be explicitly documented for an implementation either directly or indirectly by reference to a standard or publically available specification.

5.1.4.2 Chain Constraints

Chain constraints shall indicate requirements for use in the certificate path validation process as specified in ETSI TS 119 172-1 [4], clause A.4.2.1, table 6 row m.

5.1.4.3 Cryptographic Constraints

Chain constraints shall indicate requirements on algorithms and parameters used when creating signatures or used when validating signed objects as specified in ETSI TS 119 172-1 [4], clause A.4.2.1, table 6 row p.

5.1.4.4 Signature Elements Constraints

Chain constraints shall indicate requirements on the DTBS as specified in ETSI TS 119 172-1 [4], clause A.4.2.1, table 6 row b.

5.2 Basic building blocks

5.2.1 Description

This clause presents basic building blocks that are useable in the signature validation process. Later clauses use these blocks to construct validation algorithms for specific scenarios. Figure 12 shows, in a simplified way, how these building blocks are related to achieve signature validation. It closely resembles the basic validation specified in clause 5.3.

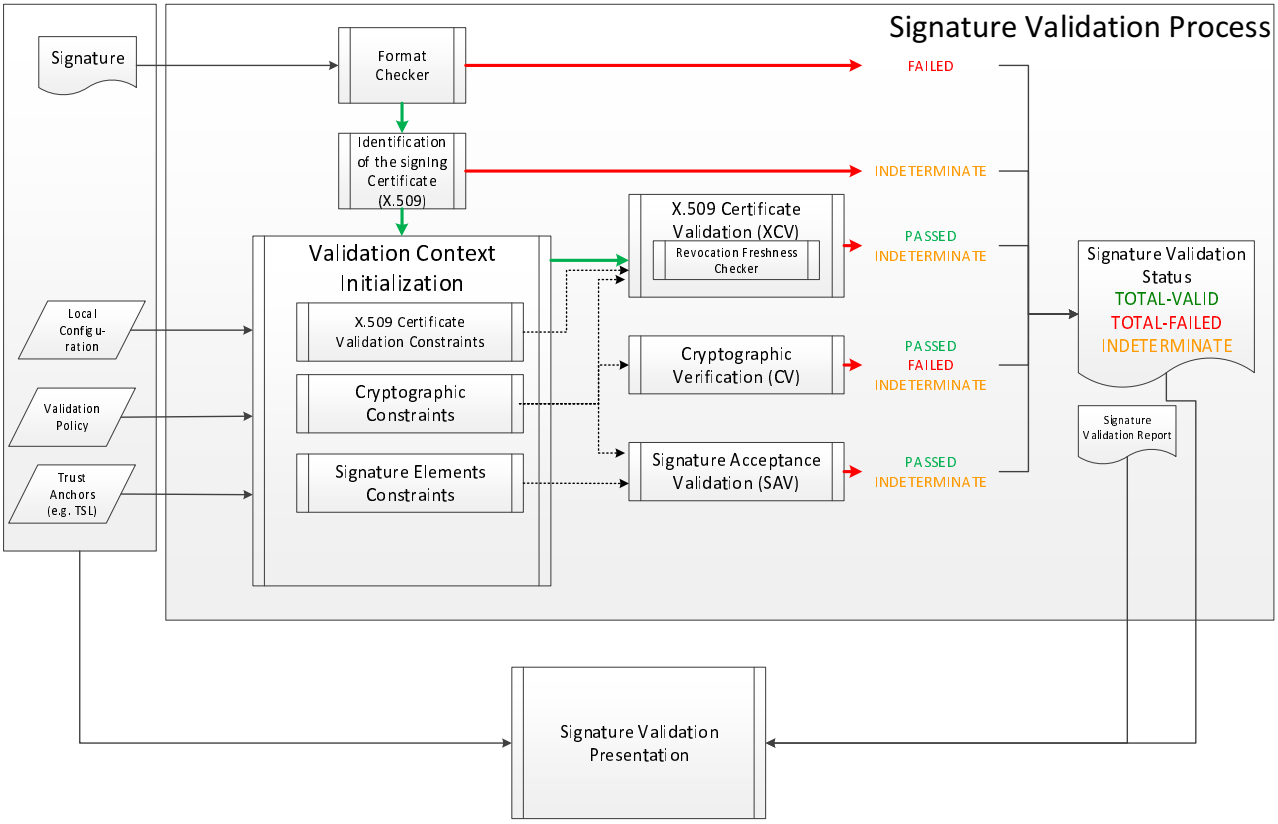

Figure 12: Signature Validation

5.2.2 Format Checking

5.2.2.1 Description

This building block shall check that the signature to validate is conformant to the applicable base format (e.g. CMS [i.8], CAdES [i.2], XML-DSig [i.11], XAdES [i.4], etc.) prior to any subsequent processing.

NOTE 1: Details on how to check format conformance 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 a specific signature.

NOTE 2: This checking process does not include any checks on conformance to a specific signature profile or a specific level of signature, like XAdES-E-XL or PAdES-LTA. Such checking, if required by the signature validation policy, can be done in the signature acceptance validation building block as specified in clause 5.2.8.

5.2.2.2 Inputs

Table 7: Inputs to the format checking building block

Input	Requirement
Signer's document OR a hash of the signer's document and signature	Mandatory

5.2.2.3 Outputs

In case the signature is conformant to the applicable format, the output shall be the indication *PASSED*. If the signature is not conformant, the output shall be *FAILED*.

5.2.3 Identification of the signing certificate

5.2.3.1 Description

This building block is responsible for identifying the signing certificate that will be used to validate the signature.

5.2.3.2 Inputs

Table 8: Inputs to the identification of the signing certificate building block

Input	Requirement
Signature	Mandatory
Signing Certificate	Optional

5.2.3.3 Outputs

- In case of success, the output shall be the signing certificate.
- In case the signing certificate cannot be identified, the output shall be the indication *INDETERMINATE* and the sub-indication *NO_SIGNING_CERTIFICATE_FOUND.*

NOTE: The process can only return INDETERMINATE in case the certificate is not contained in the signature and cannot be retrieved from an external resource pointed to by the signature reference.

5.2.3.4 Processing

The common way to unambiguously identify the signing certificate is by using a property/attribute of the signature containing a reference to it (see clause 4.2.5.2). The certificate can either be found in the signature or it can be obtained using external sources. The signing certificate can also be provided by the DA. If no certificate can be retrieved, the building block shall return the indication *INDETERMINATE* and the sub-indication *NO_SIGNING_CERTIFICATE_FOUND.*

The signing certificate shall be checked against all references present in the signature attributes, since one of these references is a reference to the signing certificate (see clause 4.2.5.2). The following steps are performed:

1) If the signature format used contains a way to directly identify the reference to the signers' certificate in the attribute, the building block shall check that the digest of the certificate referenced matches the result of digesting the signing certificate with the algorithm indicated; if they match, the building block shall return the signing certificate. Otherwise, the building block shall go to step 2.

2) The building block shall take the first reference and shall 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, the building block shall take the next element and shall repeat this step until a matching element has been found or all elements have been checked. If they do match, the building block shall continue with step 3. If the last element is reached without finding any match, the validation of this property shall be taken as failed and the building block shall return the indication *INDETERMINATE* with the sub-indication *NO_SIGNING_CERTIFICATE_FOUND.*

3) If the issuer and the serial number are additionally present in that reference, the details of the issuer's name and the serial number of the IssuerSerial element may be compared with those indicated in the signing certificate: if they do not match, an additional warning shall be returned with the output.

4) The building block shall return the signing certificate.

5.2.4 Validation context initialization

5.2.4.1 Description

This building block initializes the validation constraints (chain constraints, cryptographic constraints, signature elements constraints) and parameters (X.509 validation parameters including trust anchors, certificate validation data) that will be used to validate the signature. The constraints and parameters are initialized from any of the sources listed in clause 5.1.4.

5.2.4.2 Inputs

Table 9: Inputs to the validation context initialization building block

Input	Requirement
Signature	Mandatory
Signature Validation Policies	Optional
Trust anchor list (e.g. TSL)	Optional
Local configuration	Optional

5.2.4.3 Outputs

In case of failure, the building block shall output a status indication *INDETERMINATE* together with a subindication as defined in Table 10. Otherwise, the building block shall output the status indication passed together with the sets of constraints that shall be used in further validation as defined in Table 10.

Table 10: Output of the Validation context initialization building block

Indication	Additional Information/Sub-indication
PASSED	X.509 Validation Constraints
	Certificate Validation Data
	Chain Constraints
	Cryptographic Constraints
	Signature Elements Constraints
INDETERMINATE	*POLICY_PROCESSING_ERROR*
	SIGNATURE_POLICY_NOT_AVAILABLE

5.2.4.4 Processing

If the DA provides the SVA with a non-empty list of allowable signature validation policies ETSI TS 119 172-1 [4], this building block shall determine if the signature to be validated contains references to or the identifier of one of these policies in the signature policy attribute.

- If the signature contains one policy identifier which is part of the list of allowable signature validation policies, the SVA shall apply the policy identified by this identifier during validation.
- If the signature contains a policy identifier that is not contained in the list of allowable signature validation policies, it shall be a policy decision (local configuration) if default rules apply for the validation, or if the validation process is to be terminated.

If a formal policy has been identified, the building block shall perform the following checks. If any of these checks fail, then the building block shall return the indication *INDETERMINATE* with the sub-indication *POLICY_PROCESSING_ERROR* and with an indication that the validation failed to an invalid signature policy identifier property/attribute.

1) The building block shall access the electronic document identified by the contents of the property/attribute and containing the details of the policy; if it is not available, the building block shall return the indication *INDETERMINATE* with the sub-indication *SIGNATURE_POLICY_NOT_AVAILABLE*. If it cannot be parsed or processed for any other reason, the building block shall return the indication *INDETERMINATE* with the sub-indication *POLICY_PROCESSING_ERROR.*

2) The building block shall calculate a digest value on the result of applying the required transformations to the signature policy document referenced according to the property/attribute specifications for the signature format.

3) The building block shall 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, the building block shall 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 building block shall extract the validation constraints from the rules encoded in the validation policy. If an explicit commitment is identified within the signature, the building block shall select the rules of the signature policy corresponding to this commitment if any. If the commitment is unknown, the Verifier may select the rules dependent on other sources (e.g. the data being signed).

NOTE 1: The way used by the signature policy for presenting the rules and their description are out of the scope of the present document.

If the signature policy is implied, and stated so by the signature rules, the building block shall select the validation constraints mandated by the implicit signature policy.

NOTE 2: 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.

The building block shall return the indication *PASSED* together with the extracted validation constraints.

The processing of additional sources for initialization (e.g. local configuration) is out of the scope of the present document.

5.2.5 Revocation freshness checker

5.2.5.1 Description

This building block checks that a given revocation status information is "fresh" at a given validation time. The freshness of the revocation status information is the maximum accepted difference between the issuance time of the revocation status information and the current time. This process is used by other validation blocks when checking the revocation status of a certificate.

NOTE 1: Obtaining revocation status information issued at the current time is (in practice) impossible even with schemes providing real time revocation information. In practice, revocation status information that has been issued shortly before the current time is used and the approximation made that the information it contains is still reliable at the current time.

NOTE 2: 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 time in the past, the revocation status information is said to be "fresh" if it has been issued after the validation time (in the past) minus the considered freshness.

5.2.5.2 Inputs

Table 11: Inputs to the RFC process

Input	Requirement
Revocation status information	Mandatory
The certificate for which the revocation is being checked	Mandatory
Validation time	Mandatory
X.509 validation constraints	Mandatory

5.2.5.3 Output

The process shall output one of the following indications together with the associated validation report data as defined in Table 12.

Table 12: Output of the RFC process

Indication	
PASSED	The revocation status information is considered fresh.
FAILED	The revocation status information is not considered fresh.

5.2.5.4 Processing

1) The building block shall get the maximum accepted revocation freshness from the X.509 validation constraints for the given certificate. If the constraints do not contain a value for the maximum accepted revocation freshness and the revocation information status is a CRL or an OCSP response IETF RFC 5280 [1], IETF RFC 6960 [i.12] with a value in the `nextUpdate` field the time interval between the fields `thisUpdate` and `nextUpdate` shall be used as the value of maximum freshness. If `nextUpdate` is not set, the building block shall return with the indication *FAILED*.

NOTE: This means that if the given validation time is after the `nextUpdate` time, the revocation status information will not be considered fresh.

2) If the issuance time of the revocation information status is after the validation time minus the considered maximum freshness, the building block shall return the indication *PASSED*. Otherwise the building block shall return the indication *FAILED*.

5.2.6 X.509 certificate validation

5.2.6.1 Description

This building block validates the signing certificate at current time.

5.2.6.2 Inputs

Table 13: Inputs to the X.509 certificate validation building block

Input	Requirement
Signing certificate	Mandatory
X.509 Validation Constraints	Mandatory
Certificate Validation Data	Optional
Chain Constraints	Optional
Cryptographic Constraints	Optional
Other Certificates	Optional
Trust Anchors	Mandatory

The validation process may acquire additional certificate validation data from external sources.

5.2.6.3 Outputs

The process shall output one of the following indications together with additional information defined in Table 14.

Table 14: Output of the X.509 certificate validation building block

Indication	Additional Information/ Sub-indication
PASSED	The certificate chain used in the successful validation Any additional validation data acquired
INDETERMINATE	*NO_CERTIFICATE_CHAIN_FOUND*
	OUT_OF_BOUNDS_NO_POE
	REVOKED_NO_POE
	CRYPTO_CONSTRAINTS_FAILURE_NO_POE
	TRY_LATER content of the NEXT_UPDATE-field of the CRL
	REVOKED_CA_NO_POE
	CHAIN_CONSTRAINTS_FAILURE
	CERTIFICATE_CHAIN_GENERAL_FAILURE

5.2.6.4 Processing

1) The building block shall check that the current time is in the validity range of the signing certificate. If this constraint is not satisfied, the building block shall return the indication *INDETERMINATE* and the sub-indication *OUT_OF_BOUNDS_NO_POE*.

2) The building block shall 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 IETF RFC 5280 [1], clause 6.1, using one of the trust anchors provided in the X.509 validation constraints input:

   a) If no new chain can be built, the building block shall return the current status and the last chain built or, if no chain has been built, the indication *INDETERMINATE* with the sub-indication *NO_CERTIFICATE_CHAIN_FOUND*.

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

3) The building block shall perform validation of the prospective certificate chain 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 be following the PKIX Certification Path Validation of IETF RFC 5280 [1], clause 6.1 with the exception of the validity model. Two validity models may be supported:

   - All certificates are valid at current time (shell model); or

   - All certificates are valid at the time they were used for issuing a certificate (chain model).

   The validity model to be used may be specified as a X.509 validation constraint. If the X.509 validation constraints do not specify a validity model, the shell model shall be used.

   NOTE 1: The chain model is e.g. required to be used by law in countries like Germany.

   The validation shall include revocation checking for each certificate in the chain. Whenever the SVA is in possession of multiple applicable instances of revocation information for a certificate, the SVA shall use the instance issued latest.

   NOTE 2: This ensures that in the case of a revoked certificate the SVA does not use a CRL, which is considered fresh but does not yet contain the revocation information, whenever a fresher CRL is already available to the SVA.

   a) If the certificate path validation returns a success indication, the building block shall run the Revocation Freshness Checker (clause 5.2.5) with the used revocation status information, the certificate for which the revocation status is being checked and the current time. If the checker returns *PASSED*, the building block shall go to the next step. Otherwise, the building block shall return the indication *INDETERMINATE*, the sub-indication *TRY_LATER* and, if available, a suggestion for when to try the validation again.

b) If the certificate path validation returns a failure indication because the signing certificate has been determined to be revoked, the building block shall return the indication *INDETERMINATE*, the sub-indication *REVOKED_NO_POE*, the validated chain, the revocation date and the reason for revocation.

c) If the certificate path validation returns a failure indication because the signing certificate has been determined to be on hold, the building block shall terminate returning the indication *INDETERMINATE*, the sub-indication *TRY_LATER*, the suspension time and, if available, the content of the `nextUpdate` -field of the CRL used as the suggestion for when to try the validation again.

d) If the certificate path validation returns a failure indication because an intermediate CA is revoked, the building block shall set the current status to *INDETERMINATE/REVOKED_CA_NO_POE* and shall go to step 2. Or

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

4) The building block shall apply the Chain Constraints to the chain. Certificate constraints shall be taken into account when checking these constraints against the chain. If the chain does not match these constraints, the building block shall set the current status to *INDETERMINATE/CHAIN_CONSTRAINTS_FAILURE* and shall go to step 2.

5) The building block shall apply the cryptographic constraints to the chain. If the chain does not match these constraints, the building block shall set the current status to *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE* and shall go to step 2. And

6) The building block shall return the chain with the indication *PASSED*.

NOTE 3: Chain construction (step 2) and validation (step 3) can 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 4: For more information and rationale about certificate chain construction, refer to IETF RFC 4158 [i.1].

5.2.7 Cryptographic verification

5.2.7.1 Description

This building block checks the integrity of the signed data by performing the cryptographic verifications.

5.2.7.2 Inputs

Table 15: Inputs to the cryptographic validation building block

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

NOTE 1: In most cases, the cryptographic verification requires only the signing certificate and not the entire validated chain. However, for some algorithms the full chain can be required (e.g. the case of DSS/DSA public keys which inherit their parameters from the issuer certificate).

NOTE 2: When validating signatures like *detached signatures*, where only the hashes of objects are signed but the objects themselves are not part of the signature, it is unspecified whether it is the task of the DA to validate these hashes or whether an implementation uses the current clause to having the hash(es) of such objects validated by the SVA. Both variants are possible.

5.2.7.3 Outputs

The process shall output one of the following indications together with the associated validation report data as listed in Table 16:

Table 16: Outputs of the cryptographic validation building block

Indication		Description	Additional data items
PASSED		The signature passed the cryptographic verification.	
FAILED	*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.

5.2.7.4 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 ETSI TS 119 122-1 [i.2], ETSI TS 119 122-2 [i.3], ETSI TS 119 132-1 [i.4], ETSI TS 119 132-2 [i.5], ETSI TS 119 142-1 [i.6], ETSI TS 119 142-2 [i.7] and IETF RFC 3852 [i.8] for details.

1) The building block shall obtain the signed data object(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, the building block shall return the indication *INDETERMINATE* with the sub-indication *SIGNED_DATA_NOT_FOUND*.

2) The SVA shall check the integrity of the signed data objects. In case of failure, the building block shall return the indication *FAILED* with the sub-indication HASH_*FAILURE*.

3) The building block shall verify the cryptographic signature using the public key extracted from the signing certificate in the chain, the signature value and the signature algorithm extracted from the signature. If this cryptographic verification outputs a success indication, the building block shall return the indication *PASSED*.

4) Otherwise, the building block shall return the indication *FAILED* and the sub-indication *SIG_CRYPTO_FAILURE*.

5.2.8 Signature acceptance validation (SAV)

5.2.8.1 Description

This building block covers any additional verification to be performed on the signature itself or on the attributes of the signature ETSI TS 119 132-1 [i.4].

5.2.8.2 Inputs

Table 17: Inputs to the SAV building block

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

5.2.8.3 Outputs

The process shall output one of the following indications together with the additional information as defined in Table 18.

Table 18: Outputs of the SVA building block

Indication		Description	Additional data items
PASSED		The signature is conformant with the validation constraints.	
INDETERMINATE	*SIG_CONSTRAINTS_FAILURE*	The signature is not conformant with the validation constraints.	The set of constraints that are not verified by the signature.
	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.	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.

5.2.8.4 Processing

5.2.8.4.1 General requirements

For each constraint:

- If the constraint necessitates processing a property/attribute in the signature, the building block shall perform the processing of the property/attribute as specified in clauses 5.2.8.4.2.1 to 5.2.8.4.2.7.
- 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, the building block shall return the indication *INDETERMINATE* with the sub-indication *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 has been considered secure by the cryptographic constraints.

NOTE: This check is used when the algorithm or key size used was at the time of signing the signed object secure and only expired years later. Long term validation still allows 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, in 2013, 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.14]. 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, the building block shall return the indication *INDETERMINATE* with the sub-indication *SIG_CONSTRAINTS_FAILURE* together with the set of constraints that are not satisfied by the signature. And,
- If all the constraints are satisfied, The building block shall return the indication *PASSED.*

The building block may ignore processing a property/attribute for which no validation constraint is specified.

5.2.8.4.2 Processing AdES attributes

5.2.8.4.2.1 Processing signing certificate reference constraint

If the Signing Certificate Identifier attribute contains references to other certificates in the path, the building block shall check each of the certificates in the certification path against these references.

When this property contains one or more references to certificates other than those present in the certification path, the building block shall return the indication *INDETERMINATE* with the sub-indication *SIG_CONTRAINTS_FAILED.*

When one or more certificates in the certification path are not referenced by this property, and the signature policy mandates that references to all the certificates in the certification path have to be present, the building block shall return the indication *INDETERMINATE* with the sub-indication *SIG_CONTRAINTS_FAILED.*

5.2.8.4.2.2 Processing claimed signing time

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

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

5.2.8.4.2.3 Processing signed data object format

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

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

5.2.8.4.2.4 Processing indication of production place of the signature

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

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

5.2.8.4.2.5 Processing time-stamps on signed data objects

If the signature elements constraints contain specific constraints for time-stamps on signed data objects, i.e. the data covered by the signature, the building block shall check that they are satisfied. To do so, for each content-time-stamp attribute:

1) The building block shall 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) The building block shall 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. And

3) The building block shall apply the constraints for content-time-stamp attributes to the results returned in the previous steps. If any check fails, the building block shall return the indication *INDETERMINATE* with the sub-indication *SIG_CONSTRAINTS_FAILURE* together with an explanation of the unverified constraint.

5.2.8.4.2.6 Processing countersignatures

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

1) The building block shall perform the validation process for the countersignature in the property/attribute and the signature value octet string of the signature as the signed data object. And

2) The building block shall apply the constraints for countersignature attributes to the result returned in the previous step. If any check fails, the building block shall return the indication *INDETERMINATE* with the sub-indication *SIG_CONSTRAINTS_FAILURE* together with an explanation of the unverified constraint.

If the signature elements constraints do not contain any constraint on countersignatures, the building block 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 successfully validated.

5.2.8.4.2.7 Processing signer attributes

If the signature elements constraints define specific constraints for certified attributes and signed assertions,

1) The building block shall validate the attribute certificate(s) and signed assertions present in this property/attribute following the rules established in ISO/IEC 9594-8 [2].

2) The building block shall check that the attribute certificate(s) and signed assertions actually match the rules specified in the input constraints. And

3) If any check fails, the building block shall return the indication *INDETERMINATE* with the sub-indication *SIG_CONSTRAINTS_FAILURE* with more information on the constraint that could not be verified.

If the signature rules do not specify rules for certified attributes or signed assertions, the SVA shall make the value of such attribute or signed assertions available to its DA so that it can decide on additional suitable processing, which is out of the scope of the present document.

5.2.9 Signature validation presentation building block

The Signature Validation Presentation is an optional element in the signature validation process that can be used by a verifier to check the results of a validation process. When present, the signature validation presentation building block should support:

- Presenting the data (SD) that has been covered by the signature.
- Presenting information identifying the signer.
- Presenting 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 (*TOTAL-PASSED*, *TOTAL-FAILED*, *INDETERMINATE*).
- In case of *TOTAL-FAILED*: Presenting the reason for the signature being invalid.
- In case of *INDETERMINATE*: Highlighting the parts of the validation report that indicates steps to be taken to potentially get to a determinate result. And
- Presenting the validation report.

5.3 Validation process for Basic Signatures

5.3.1 Description

This clause describes a validation process for validating Basic Signatures as per 4.3.2. This process itself will also be used as a building block by the validation process of time-stamps (see clause 5.4) and of Signatures with Time (see clause 5.5). The process is built on the building blocks described in the previous clause.

5.3.2 Inputs

Table 19: Inputs to Basic Validation

Input	Requirement
Signature	Mandatory
Signed data object (s) or hash of SDO(s)	Optional
Signing Certificate	Optional
Trust anchor list (e.g. TSL)	Optional
Signature Validation Policies	Optional
Local configuration	Optional
Certificate Validation Data	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 5.1.3).

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.

1) The Basic Signature validation process shall perform the format checking as per clause 5.2.2. If the process returns *PASSED*, the Basic Signature validation process shall continue with the next step. Otherwise the Basic Signature validation process shall return the indication *INDETERMINATE* with the sub-indication *FORMAT_FAILURE*.

2) The Basic Signature validation process shall perform the identification of the signing certificate (as per clause 5.2.3) with the signature and the signing certificate, if provided as a parameter. If the identification of the signing certificate process returns the indication *INDETERMINATE* with the sub-indication *NO_SIGNING_CERTIFICATE_FOUND*, the Basic Signature validation process shall return the indication *INDETERMINATE* with the sub-indication *NO_SIGNING_CERTIFICATE_FOUND*, otherwise it shall go to the next step.

3) The Basic Signature validation process shall perform the Validation Context Initialization as per clause 5.2.4. If the process returns *INDETERMINATE* with some sub-indication, return with the indication *INDETERMINATE* together with that sub-indication, otherwise go to the next step.

4) The Basic Signature validation process shall perform the Cryptographic Verification process as per clause 5.2.7 with the following inputs:

   a) The signature.

   b) The certificate chain returned in the previous step. And

   c) The signed data object(s).

   If the cryptographic signature validation process returns PASSED, the Basic Signature validation process shall go to the next step. Otherwise, the Basic Signature validation process shall return the returned indication, sub-indication and associated information provided by the cryptographic signature validation process.

5) The Basic Signature validation process shall perform the X.509 Certificate Validation as per clause 5.2.6 with the following inputs:

   a) The signing certificate obtained in step 1. And

   b) X.509 validation constraints, certificate validation-data, chain constraints and cryptographic constraints obtained in step 3 or provided as input.

   If the signing certificate validation process returns the indication *PASSED*, the Basic Signature validation process shall go to the next step.

If the signing certificate validation process returns the indication *INDETERMINATE* with the sub-indication *REVOKED_NO_POE* and if the signature contains a content-time-stamp attribute, the Basic Signature validation process shall perform the validation process for AdES time-stamps as defined in clause 5.4. If this process returns the indication *PASSED* and the generation time of the time-stamp token is after the revocation time, the Basic Signature validation process shall return the indication *FAILED* with the sub-indication *REVOKED*. In all other cases, the Basic Signature validation process shall return the indication *INDETERMINATE* with the sub-indication *REVOKED_NO_POE*.

If the signing certificate validation process returns the indication *INDETERMINATE* with the sub-indication *OUT_OF_BOUNDS_NO_POE* and if the signature contains a content-time-stamp attribute, the Basic Signature validation process shall perform the validation process for AdES time-stamps as defined in clause 5.4. If it returns the indication *PASSED* and the generation time of the time-stamp token is after the expiration date of the signing certificate, the Basic Signature validation process shall return the indication *INDETERMINATE* with the sub-indication *EXPIRED*. Otherwise, the Basic Signature validation process shall return the indication *INDETERMINATE* with the sub-indication *OUT_OF_BOUNDS_NO_POE*.

In all other cases, the Basic Signature validation process shall return the indication, sub-indication and any associated information returned by the signing certificate validation process.

6) The Basic Signature validation process shall perform the Signature Acceptance Validation process as per clause 5.2.8 with the following inputs:

   a) The signature.

   b) The Cryptographic Constraints. And

   c) The Signature Elements Constraints.

   If the signature acceptance validation process returns *PASSED*, the Basic Signature validation process shall go to the next step.

   If the signature acceptance validation process returns the indication *INDETERMINATE* with the sub-indication *CRYPTO_CONSTRAINTS_FAILURE_NO_POE* and the material concerned by this failure is the signature value and if the signature contains a content-time-stamp attribute, the Basic Signature validation process shall perform the validation process for AdES time-stamps as defined in clause 5.4. If it returns the indication *PASSED* and the algorithm(s) concerned were no longer considered reliable at the generation time of the time-stamp token, the Basic Signature validation process shall return the indication *FAILED* with the sub-indication *CRYPTO_CONSTRAINTS_FAILURE*. In all other cases, the Basic Signature validation process shall return the indication *INDETERMINATE* with the sub-indication *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*, the validation process for signature with long-term validation data and with archival data can be used to validate the signature, if other POE (e.g. from a trusted archive) exist.

   In all other cases, the Basic Signature validation process shall return the indication and associated information returned by the signature acceptance validation building block.

7) The Basic Signature validation process shall return the success indication *PASSED*.
   In addition, the Basic Signature validation process 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 attributes, including those which were not processed during the validation process.

NOTE 4: What the DA does 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 building block for the validation of an IETF RFC 3161 [3] or ETSI TS 119 422 [i.13] time-stamp token.

An IETF RFC 3161 [3] or ETSI TS 119 422 [i.13] time-stamp token is a Basic Signature. Hence, the validation process builds on the validation process of a Basic Signature.

5.4.2 Inputs

Table 20: Inputs to time-stamp validation

Input	Requirement
Time-stamp token	Mandatory
Trust anchor list (e.g. TSL)	Optional
Signature Validation Policies	Optional
Local configuration	Optional
Time-Stamp Certificate	Optional

5.4.3 Outputs

The main output of the time-stamp validation is a status indicating the validity of the time-stamp. This status may be accompanied by additional information (see clause 5.1.3).

5.4.4 Processing

1) Token signature validation: the building block shall perform the validation process for Basic Signatures as per clause 5.3 with the time-stamp token. In all the steps of this process, the building block shall take into account that the signature to validate is a time-stamp token (e.g. to select TSA trust-anchors). If this step returns *PASSED*, the building block shall go to the next step. Otherwise, the building block shall return the indication and information returned by the validation process.

2) Data extraction: in addition to the data items returned in step 1, the building block:

   - shall return the generation time and the message imprint present in the `TSTInfo` field of the time-stamp token; and
   - may return other data items present in the `TSTInfo` field of the time-stamp token.

These items may be used by the building block in the process of validating the AdES signature.

5.5 Validation process for Signatures with Time and Signatures with Long-Term Validation Data

5.5.1 Description

This clause describes a validation process for Signatures with Time as defined in clause 4.3.3 and Signatures with Long-Term Validation Data. Signatures with Long-Term Validation Data differ from Signatures with Time by the fact that they contain additional validation material that can be used during validation. The validation processes are identical.

5.5.2 Inputs

Table 21: Inputs to validation of signatures with time

Input	Requirement
Signature	Mandatory
Signed data object (s)	Optional
Trust anchor list (e.g. TSL)	Optional
Signature Validation Policies	Optional
Local configuration	Optional
Signing 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 5.1.3).

5.5.4 Processing

1) The process shall initialize the set of signature time-stamp tokens from the signature time-stamp attributes present in the signature and shall 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: the process shall perform the validation process for Basic Signatures as per clause 5.3 with all the inputs, including the processing of any signed attributes as specified. If the Signature contains long-term validation data, this data shall be passed to the validation process for Basic Signatures.

   If this validation returns *PASSED, INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE, INDETERMINATE/REVOKED_NO_POE or INDETERMINATE/OUT_OF_BOUNDS_NO_POE*, the SVA shall go to the next step. Otherwise, the process shall return the status and information returned by the validation process for Basic Signatures.

NOTE 2: the process in the case *INDETERMINATE/REVOKED_NO_POE* is continued, because a proof that the signing occurred before the revocation date can help to go from *INDETERMINATE* to *TOTAL-PASSED* (step 4-a).

NOTE 3: the process in the case *INDETERMINATE/OUT_OF_BOUNDS_NO_POE* is continued, because a proof that the signing occurred before the issuance date (notBefore) of the signing certificate can help to go from *INDETERMINATE* to *TOTAL-FAILED* (step 4-b).

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

3) Signature time-stamp validation:

   a) For each time-stamp token in the set of signature time-stamp tokens, the process shall check that the message imprint has been generated according to the corresponding signature format specification verification. If the verification fails, the process shall 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 process shall perform the time-stamp validation process as per clause 5.4:

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

      In all other cases:

      - If no specific constraints mandating the validity of the attribute are specified in the validation constraints, the process shall remove the time-stamp token from the set of signature time-stamp tokens and shall try the next token.

      - Otherwise, the process shall return the indication/sub-indication and associated explanations returned from the Time-stamp token validation process.

4) Comparing times:

   a) If step 2 returned the indication *INDETERMINATE* with the sub-indication *REVOKED_NO_POE:* If the returned revocation time is posterior to best-signature-time, the process shall perform step 4d. Otherwise, the process shall return the indication *INDETERMINATE* with the sub-indication *REVOKED_NO_POE*.

b) If step 2 returned the indication *INDETERMINATE* with the sub-indication *OUT_OF_BOUNDS_NO_POE:* If best-signature-time is before the issuance date of the signing certificate, the process shall return the indication *FAILED* with the sub-indication *NOT_YET_VALID*. Otherwise, the process shall return the indication *INDETERMINATE* with the sub-indication *OUT_OF_BOUNDS_NO_POE*.

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

d) For each time-stamp token remaining in the set of signature time-stamp tokens, the process shall 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 computed on the signed data. The process shall apply the rules specified in IETF RFC 3161 [3], 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 validation constraints. If all the checks end successfully, the process shall go to the next step. Otherwise the process shall return the indication *FAILED* with the sub-indication *TIMESTAMP_ORDER_FAILURE*.

5) Handling Time-stamp delay: If the validation constraints specify a time-stamp delay:

a) If no signing-time property/attribute is present, the process shall return the indication *INDETERMINATE* with the sub-indication *SIG_CONSTRAINTS_FAILURE*.

b) If a signing-time property/attribute is present, the process shall check that the claimed time in the attribute plus the time-stamp delay is after the best-signature-time. If the check is successful, the process shall go to the next step. Otherwise, the process shall return the indication *FAILED* with the sub-indication *SIG_CONSTRAINTS_FAILURE*.

6) Data extraction: the process shall return the success indication *PASSED.*
In addition, the process should return additional information extracted from the signature and/or used by the intermediate steps.
In particular, the process should return intermediate results such as the validation results of any signature time-stamp token.

NOTE 5: What the DA does with this information is out of the scope of the present document.

NOTE 6: In the algorithm above, the signature-time-stamp protects the signature against the revocation of the signing certificate (step 4-a) but not against expiration. The latter case can require validating the signing certificate in the past (see clause 5.6) because not all CAs provide revocation data for expired certificates or are willing to revoke certificates after expiration.

5.6 Validation process for Signatures with Archival Data

5.6.1 Introduction

This clause describes a validation process for Signatures with Archival Data.

The process described in this clause may also be used to validate Basic Signatures.

NOTE 1: This is in particular useful in the case where the SVA takes 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-assertion).

NOTE 2: Such a validation can be done off-line when all required validation material is available within the signature and local configuration.

The process is built on the building blocks described in clause 5.2 and the additional building blocks defined in clause 5.6.2.

5.6.2 Additional building blocks

5.6.2.1 Past certificate validation

5.6.2.1.1 Description

This process validates a certificate at a date/time which can be in the past. This may become necessary in the long term validation settings when a compromising event (for instance, the end-entity certificate expires) prevents the traditional certificate validation algorithm (see clause 5.2.6) 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 the indication *INDETERMINATE* with the sub-indication *OUT_OF_BOUNDS_NO_POE* due to the step 1 of the processing in clause 5.2.6).

The rationale of the algorithm described below 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 (respectively the revocation status information) is proven to having existed at a date in the past when the issuer of the certificate (respectively the revocation status information) was still considered reliable and under control of its signing key.

The past certificate validation process will slide the validation time from the current time to some date in the past each time it encounters a certificate proven to be revoked (see the Validation Time Sliding process in clause 5.6.2.2). In addition to the certificate chain, the process outputs the last value of validation 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 validation time can be accepted as valid. This assertion is the basis of the LTV validation processes presented in the next clauses.

5.6.2.1.2 Input

Table 22: Inputs to past certificate validation building block

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

5.6.2.1.3 Output

Table 23: Outputs of past certificate validation building block

Indication	
PASSED + validation time + certificate chain	
INDETERMINATE	*NO_CERTIFICATE_CHAIN_FOUND*
	NO_POE
FAILED	*CHAIN_CONSTRAINTS_FAILURE*

5.6.2.1.4 Processing

1) The building block shall 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 IETF RFC 5280 [1], clause 6.1, using one of the trust anchors provided in the inputs:

   a) If no new chain can be built, the building block shall return the current status and the last chain built or, if no chain was built, the indication *INDETERMINATE* with the sub-indication *NO_CERTIFICATE_CHAIN_FOUND*.

   b) Otherwise, the building block shall go to the next step.

2) The building block shall run the Certification Path Validation of IETF RFC 5280 [1], 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 for the signing certificate:

   a) If the certificate path validation returns *PASSED*, the building block shall 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, the building block shall set the current status to *INDETERMINATE/REVOKED_CA_NO_POE* and shall go to step 1.

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

   d) If the certificate path validation returns any other failure indication, the building block shall go to step 1.

3) The building block shall perform the validation time sliding process as per clause 5.6.2.2 with the following inputs: the prospective chain, the set of POEs and the cryptographic constraints. If it outputs a success indication, the building block shall go to the next step. Otherwise, the building block shall set the current status to the returned indication and sub-indication and shall go back to step 1.

4) The building block shall apply the chain constraints to the chain. If the chain does not match these constraints, the building block shall set the current status to *FAILED/CHAIN_CONSTRAINTS_FAILUR*E and shall go to step 1.

5) The building block shall return the current status . If the current status is *PASSED*, the building block shall also return the certificate chain as well as the calculated validation time returned in step 3.

5.6.2.2 Validation time sliding process

5.6.2.2.1 Description

This building block slides the validation time from the current-time to some date in the past each time it encounters a certificate proven to be revoked.

The process outputs the last value of validation time associated with the target certificate (the certificate to validate) which is a point in time when all the certificates in the prospective certificate chain were valid.

5.6.2.2.2 Input

Table 24: Inputs to the validation time sliding building block

Input	Requirement
A prospective certificate chain	Mandatory
A set of POEs	Mandatory
Cryptographic constraints	Optional
X.509 validation constraints	Optional

5.6.2.2.3 Output

Table 25: Outputs of the validation time sliding building block

Indication	
PASSED + validation time	
INDETERMINATE	*NO_POE*

5.6.2.2.4 Processing

1) The building block shall initialize control-time to the current date/time.

   NOTE 1: Control-time is an internal variable that is used within the algorithms and not part of the core results of the validation process.

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

   a) The building block shall 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. In the case of a CRL, it shall satisfy the checks specified in IETF RFC 5280 [1] clause 6.3; and
      - The issuance date of the revocation status information is before control-time.

      If more than one revocation status is found, the building block shall consider the most recent one and shall go to the next step. If there is no such information, The building block shall return the indication *INDETERMINATE* with the sub-indication *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, the building block shall go to step c). Otherwise, the building block shall return the indication *INDETERMINATE* with the sub-indication *NO_POE*.

   c) The update of the value of control-time is as follows:

      - If the certificate is marked as revoked in the revocation status information, the building block shall set control-time to the revocation time.
      - If the certificate is not marked as revoked, the building block shall run the Revocation Freshness Checker with the used revocation information status, the certificate for which the revocation status is being checked and the control-time. If it returns *FAILED*, the building block shall set control-time to the issuance time of the revocation status information. Otherwise, the building block shall not change the value of control-time.

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

7) The building block shall continue with the next certificate in the chain or, if no further certificate exists, the building block shall return the status indication *PASSED* and the calculated control-time.

NOTE 2: 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 3: The rationale 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 it issues revocation information status for expired certificates (for instance, using the CRL extension expiredCertOnCRL).

NOTE 4: 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 rationale of sliding control-time in step 2-d).

NOTE 5: 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 to be compared to the control-time associated to the certificate.

NOTE 6: When all the certificates in the chain can be validated at the current time, the control-time never slides and the current time is returned.

5.6.2.3 POE extraction

5.6.2.3.1 Description

This building block derives POEs from a given time-stamp. Assumptions:

- The time-stamp validation has returned *PASSED*.
- 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 time-stamp 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.

EXAMPLE: A time-stamp on the signature value gives a POE of the signature value at the generation date/time of the time-stamp.

A time-stamp can also give an indirect POE when it is computed on the hash value of some data instead of the data itself. A POE for *DATA* at T_1 can be derived from the time-stamp:

- If there is a POE for h(*DATA*) at a date T_1,where h is a cryptographic hash function and *DATA* is some data (e.g. a certificate),
- if h is asserted in the cryptographic constraints to be trusted until at least a date T after T_1; and
- if there is a POE for *DATA* at a date T after T_1.

5.6.2.3.2 Input

Table 26: Inputs to the POE extraction building block

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

5.6.2.3.3 Output

The POE extraction process shall return a set of POEs that may be an empty set.

5.6.2.3.4 Processing

1) The building block shall determine the set *S* of references to objects and objects that are part of the signature and are protected by the time-stamp.

2) If any of the objects in the set *S* contains other objects that are not yet contained in the set S and that can be used in signature validation, the building block shall add them to the set S.

EXAMPLE: Such objects can be a PAdES Document Security Store or signed data like a PAdES Signed Data element.

3) The building block shall initialize the set *P* of POE with an empty set.

4) For each reference to objects contained in the set *S* where the reference contains a hash value of the referenced object O and the cryptographic hash function h is asserted in the cryptographic constraints to be trusted until at least a date after the time of the generation of the timestamp (named T1):

   a) The building block shall add to *P* a POE for the hash value h(O) of the object O at T1.

   b) If the set of POEs includes a POE for an object O at a date/time T after T1, the building block shall add to *P* a POE for O at T1.

5) For each object contained in *S*, the building block shall add to *P* a POE for that object at T1.

6) The building block shall return the set *P* of POEs.

5.6.2.4 Past signature validation building block

5.6.2.4.1 Description

This building block 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.

5.6.2.4.2 Input

Table 27: Inputs to the past signature validation building block

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

5.6.2.4.3 Output

This process shall output an indication/sub-indication, which is either the same as the current time indication/sub-indication given in the inputs or one of the following: *PASSED, FAILED or INDETERMINATE/NOT_YET_VALID.*

5.6.2.4.4 Processing

1) The building block shall perform the past certificate validation process with the following inputs: the signature, the target certificate, the X.509 validation parameters, certificate validation data, chain constraints, cryptographic constraints and the set of POEs. If it returns *PASSED*/*validation time*, the building block shall go to the next step. Otherwise, the building block shall return the current time status and sub-indication with an explanation of the failure.

2) If there is a POE of the signature value at (or before) the validation time returned in the previous step:

   - If current time indication/sub-indication is *INDETERMINATE/REVOKED_NO_POE* or *INDETERMINATE/ REVOKED_CA_NO_POE*, the building block shall return *PASSED*.

   - If current time indication/sub-indication is *INDETERMINATE/OUT_OF_BOUNDS_NO_POE*:

     a) If best-signature-time (lowest time at which there exists a POE for the signature value in the set of POEs) is before the issuance date of the signing certificate (notBefore field), the building block shall return the indication *INDETERMINATE* with the sub-indication NOT_YET_VALID.

b) If best-signature-time (lowest time at which there exists a POE for the signature value in the set of POEs) is after the issuance date and before the expiration date of the signing certificate, the building block shall return the status indication *PASSED*.

3) If current time indication/ sub-indication 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 the time up to which the algorithm in question was considered secure, the building block shall return the status indication *PASSED*.

4) In all other cases, the building block shall return the current time indication/ sub-indication together with an explanation of the failure.

5.6.2.5 Evidence record validation building block

5.6.2.5.1 Description

This process is used to validate an Evidence Record as specified in IETF RFC 4998 [i.9] or IETF RFC 6283 [i.10].

An Evidence Record proves that an archive object existed and has not been changed from the time of the initial Time-Stamp Token within the first archive time-stamp (ATS). In order to complete the non-repudiation proof for an archive object, the last ATS has to be valid and archive time-stamp chains and their relations to each other have to be proven.

5.6.2.5.2 Input

Table 28: Inputs to the evidence record validation building block

Input	Requirement
Signed data object or group of data objects	Mandatory
Evidence Record(s)	Mandatory
Cryptographic constraints	Optional
Trust anchor list (e.g. TSL)	Optional
Signature Validation Policies	Optional
Local configuration	Optional
Time-Stamp Certificate	Optional

5.6.2.5.3 Output

This process shall output one of the following status codes: *PASSED* or *FAILED*.

5.6.2.5.4 Processing

1) Verify that the first Archive Time-stamp of the first Archive Time-stamp Chain (the initial Archive Time-stamp) of the Evidence Record contains the hash value of the data object or data object group according to the EncapsulatedContentInfo of the Signed Data object (group). If this is the case, the building block shall go to the next step. Otherwise, the building block shall return the indication FAILED.

2) The building block shall verify each Archive Time-stamp Chain:

   a) The building block shall check that the first hash value list of each Archive Time-stamp **(except the initial Archive Time-stamp)** contains the hash value of the Time-stamp of the previous Archive Time-stamp. If this is the case, the building block shall go to the next step. Otherwise, the building block shall return the indication *FAILED*.

   b) Performing the time stamp validation process (see clause 5.4) and if necessary, the past signature validation process (see clause 5.6.2.4):

      b1) The building block shall check that each Archive Time-stamp is valid relative to the time of the following Archive Time-stamp. If this is the case, the building block shall go to the next step. Otherwise, the building block shall return the indication *FAILED*.

b2) The building block shall check that all Archive Time-stamps within a chain use the same hash algorithm and this algorithm is considered secure at the time of the first Archive Time-stamp of the following Archive Time-stamp Chain. If this is the case, the building block shall go to the next step. Otherwise, the building block shall return the indication *FAILED*.

3) The building block shall verify that the first hash value list (partialHashtree) of the first Archive Time-stamp of all other Archive Time-stamp Chains contains a hash value of the concatenation of the data object hash and the hash value of all older Archive Time-stamp Chain. If this is the case, the building block shall go to the next step. Otherwise, the building block shall return the indication *FAILED*.

4) The building block shall verify that each Archive Time-stamp was generated before the last Archive Time-stamp of the preceding Archive Time-stamp Chain became invalid. If this is the case, the building block shall go to the next step. Otherwise, the building block shall return the indication *FAILED*.

5) The building block shall verify the last Archive Time-Stamp using the validation process for time-stamps (see clause 5.4). If the process returns *PASSED*, return with the indication *PASSED*.

6) Otherwise, return with the indication *FAILED*.

5.6.3 Long term validation process

5.6.3.1 Description

This process is used when validation signatures with Long-Term Validation Data or signatures with Archival Data. Several unsigned attributes can be present in both signatures classes:

- Time-stamp(s) on the signature value (Signature with Time);
- Time-stamp(s) on the references of validation data;
- Time-stamp(s) on the references of validation data, the signature value and the signature time-stamp;
- Attributes with the values of validation data; or
- Attributes with references to validation data.

In addition, Signatures with Archival Data contain the following attributes:

- Archive time-stamp(s) on the whole signature except the last archive time-stamp; or
- Evidence Records on part or the whole signature.

The process described in this clause validates any of the classes of signatures.

The DA may provide to the SVA an initial set of POEs proving the existence of elements used in validation, like e.g. the signature, certificates or time-stamps. The structure or format of these POEs are implementation dependent.

5.6.3.2 Input

Table 29: Inputs to the long term validation process

Input	Requirement
Signature	Mandatory
Signed data object (s)	Optional
Trust anchor list (e.g. TSL)	Optional
Signature Validation Policies	Optional
Local configuration	Optional
A set of POEs	Optional
Signing Certificate	Optional
Evidence Records	Optional

5.6.3.3 Output

The main output of this signature validation process shall be a status indicating the validity of the signature. This status may be accompanied by additional information (see clause 5.1.3).

5.6.3.4 Processing

1) If there is one or more evidence records, the long term validation process shall perform the evidence record validation process for each of them according to clause 5.6.2.5. If the evidence record validation process returns PASSED, the SVA shall go to step 6.

2) POE initialization: the long term validation process shall 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.

3) The long term validation process shall perform the validation process for Signatures with Time as per clause 5.5 with all the inputs, including the processing of any signed attributes as specified.

- If the validation outputs *PASSED*:
  - If there is no validation constraint mandating the validation of the LTV attributes, the long term validation process shall return the indication *PASSED*.
  - Otherwise, the SVA shall go to step 4.
- If the validation outputs one of the following indications/sub-indications: INDETERMINATE/REVOKED_NO_POE, INDETERMINATE/REVOKED_CA_NO_POE, INDETERMINATE/OUT_OF_BOUNDS_NO_POE or INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE, the long term validation process shall go to the next step.
- In all other cases, the long term validation process shall fail with returned code and information.

NOTE 2: long term validation is done 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 can help to go from INDETERMINATE to a determined status.

NOTE 3: Performing the long term validation part of the algorithm even when the basic validation returns *PASSED* can be useful in the case the SVA is controlled by an archiving service. In such cases, it can be necessary to ensure that any long term attribute present in the signature is actually valid before making a decision about the archival of the signature.

NOTE 4: Steps 3 and 4 below are not part of the validation process per se, but are present to collect PoEs for step 5.

4) If there is at least one time-stamp attribute:

   a) The long term validation process shall select the newest time-stamp that has not been processed and perform the time-stamp validation, as per clause 5.4.

   b) If *PASSED* 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, the long term validation process shall perform the POE extraction process with the signature, the time-stamp and the cryptographic constraints as inputs. The long term validation process shall add the returned POEs to the set of POEs.

   c) Otherwise, the long term validation process shall perform past signature validation process with the following inputs: the time-stamp, the indication/sub-indication returned by the time-stamp validation process, the TSA's certificate, the X.509 validation parameters, X.509 validation constraints, cryptographic constraints and the set of POEs.

- If it returns *PASSED* and the cryptographic hash function used in the time-stamp is considered reliable at the generation time of the time-stamp, the long term validation process shall perform the POE extraction process and shall add the returned POEs to the set of POEs continue with step 4 using the next time-stamp attribute.

- In all other cases:

  - If no specific constraints mandating the validity of the attribute are specified in the validation constraints, the long term validation process shall ignore the attribute and shall continue with step 4 using the next time-stamp attribute.

  - Otherwise, the long term validation process shall fail with the returned indication/sub-indication and associated explanations.

d) If all time-stamp attributes have been processed, continue with step 5. Otherwise, continue with step 4b.

5) Past signature validation: the long term validation process shall perform the past signature validation process with the following inputs: the signature, the status indication/sub-indication returned in step 2, the signing certificate, the X.509 validation parameters, certificate validation data, chain constraints, cryptographic constraints and the set of POEs. If it returns *PASSED* the long term validation process shall go to the next step. Otherwise, the long term validation process shall return the indication/sub-indication and associated explanations returned from the past signature validation process.

6) Data extraction: the SVA shall return the success indication *PASSED.* In addition, the long term validation process should return additional information extracted from the signature and/or used by the intermediate steps. In particular, the long term validation process should return intermediate results such as the validation results of any time-stamp token.

NOTE 5: What the DA does with this information is out of the scope of the present document.

Annex A (informative): Validation examples

A.1 General remarks and assumptions

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.

- While validating a signature with time 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 long term validation process also for signature with time. In the examples this distinction is ignored and only the logic behind the algorithm is presented as applicable to the examples chosen.
- These examples also assume that basic checks like cryptographic or format checks succeed. The focus is 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, the following assumptions are made:
  - The signing certificate can be identified, as it is provided within the signature.
  - There are no specific constraints on the validation process unless noted otherwise.
  - A valid path to a trust anchor can be built for all certificates used unless noted otherwise.
  - Only the signature is needed as an input unless noted otherwise.
  - The syntax/format of all elements is correct.
  - All required elements are present.
  - Time-stamps and signatures have been calculated over the right data.
  - No other similar basic flaws exist, unless noted otherwise.

A.2 Symbols

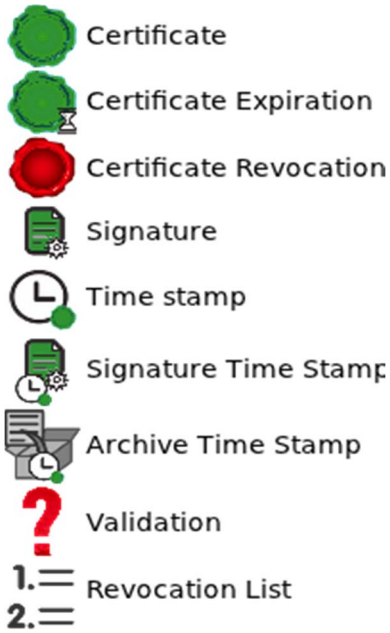

Figure A.1: Symbols used in examples

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

A.3 Example 1: Revoked certificate

A.3.1 Introduction

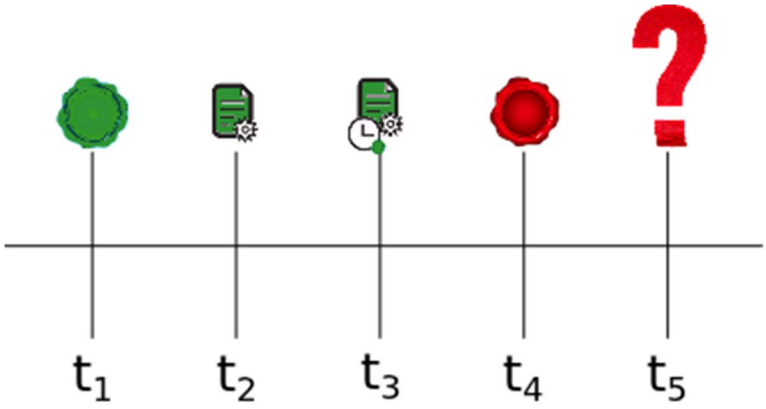

Figure A.2: Revoked Certificate Example

In this example a simple case is shown where a certificate is revoked before subsequent validation of a signature. Figure A.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 time-stamp is created (*Signature with Time)*.
- At time t4 the certificate is revoked.
- At time t5 validation of the certificate is tried.
- All other certificates that are used in the process are assumed to being still valid.

A.3.2 Basic signature validation

Expected result	*INDETERMINATE*/REVOKED_NO_POE
Rationale	The Basic Signature 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.

The validation algorithm defined in clause 5.3 proceeds as follows:

- The identification of the signing certificate succeeds by assumption.
- The initialization of the validation constraints and parameters succeeds by assumption.
- The validation of the signing certificate returns *INDETERMINATE*/*REVOKED_NO_POE* since the signing certificate has been revoked.

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

A.3.3 Validating a signature with time

Expected result	*TOTAL-PASSED*
Rationale	The status goes from *INDETERMINATE/REVOKED_NO_POE* (using the basic validation algorithm) to *TOTAL-PASSED* because the signature with time validation algorithm processes the signature time-stamp attribute and finds that the signing time lies before the revocation date.

The validation algorithm for signatures with time defined in clause 5.5 proceeds as follows:

- The set of signature time-stamp tokens is initialized to the single time-stamp present in the signature (step 1).
- Best-signature-time is set to current time (step 1).
- The Basic Signature validation is performed . As shown before, this returns *INDETERMINATE/REVOKED_NO_POE*, and the rest of the algorithm can be run, since existing time-stamps can still allow us to verify the signature.
- The verification (step 3-a) of the message imprint of the time-stamp succeeds by assumption.
- The Time-stamp token validation (step 3-b) is performed as per clause 5.4 for verifying the time-stamp.
- The Basic Signature validation of the signature on the time-stamp token succeeds, since the certificate of the TSA has neither expired nor been revoked by assumption.
- Since the previous step returned *TOTAL-PASSED*, the signature has been created before the time-stamp and the best-signature-time is set to the time of the time-stamp (step 4-b).
- Step 4-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, the process continues with step 4-d.
- The coherence of the time values is checked and found to be ok (step 4-c).
- No constraints on time-stamp delay exist (step 5), so the process skips to the next step.
- The process returns *TOTAL-PASSED* and returns the validation report generated to the DA (step 6).

A.3.4 Example 2: Revoked CA certificate

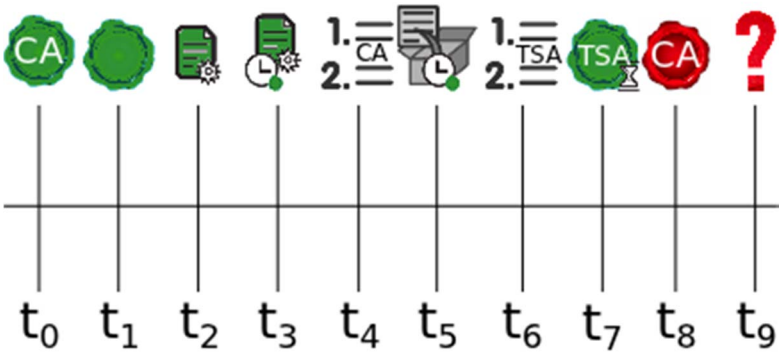

Figure A.3: Revoked CA Certificate

This is a slightly more complex case, where the CA certificate that issued the signing certificate has been revoked. Figure A.3 shows the timeline for the relevant events:

- At time t0 the CA certificate is issued by another CA.
- At time t1 the signing certificate is issued by that CA.
- At time t2 the signature is created using the certificate.

- At time t3 a signature time-stamp is created (*signature with time*).
- At time t4 CRLs were issued by the CA that issued the signing certificate.
- At time t5 a Signature with Archival Data is created and an archive time-stamp 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 validation of the certificate is tried.
- All other certificates that are used in the process are assumed to being still valid.

It is assumed that the TSA certificate has been issued by a different authority than the CA certificate.

A.3.5 Basic signature validation

Expected result	*INDETERMINATE/REVOKED_CA_NO_POE*
Rationale	The algorithm for Basic Signatures does not handle the LTV attributes.

The validation algorithm defined in clause 5.3 proceeds as follows:

- The identification of the signing certificate succeeds by assumption.
- The initialize of the validation constraints and parameters succeeds by assumption.
- The validation of the signing certificate returns *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.

A.3.6 Validation of a signature with time

Expected result	*INDETERMINATE/REVOKED_CA_NO_POE*
Rationale	The algorithm for signatures with time 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.

The validation process defined in clause 5.5 proceeds as follows:

- The set of signature time-stamp tokens is initialized to the single signature time-stamp token present in the signature.
- Best-signature-time is set to current time.
- The validation process for Basic Signatures is performed and returns *INDETERMINATE/REVOKED_CA_NO_POE.*
- Since the signature validation did not report *TOTAL-PASSED* nor INDETERMINATE/REVOKED_NO_POE nor *INDETERMINATE/OUT_OF_BOUNDS*, the algorithm terminates with *INDETERMINATE/REVOKED_CA_NO_POE.*

A.3.7 Long-Term-Validation

The Long-Term-Validation-Algorithm is applied.

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

The process starts in clause 5.6.3:

- *POE initialization (step 1)*: the POE is initialized with all objects:

Content	Exists at time
The signature	T9
The signing certificate (and other certificates required to form a chain to a trust anchor)	T9
Revocation Information for the signing 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

- There is no evidence record, so step 1 is skipped.
- A first set of POEs is created using all the objects in the signature.
- *The validation process for Signatures with time* returns *INDETERMINATE/REVOKED_CA_NO_POE*. The process continues as existing time-stamps can still allow to verify the signature.
- The Time-stamp token validation (step 4) is performed as per clause 5.4 for verifying the archive time-stamp:
  - Basic signature validation of the signature on the archive time-stamp token succeeds, since the certificate of the TSA that has produced that time-stamp token has neither expired nor been revoked.
- POEs are extracted at the time of the archive time-stamp (see clause 5.6.2.3) for:
  - The signature
  - The signing certificate (and other certificates required to form a chain to a trust anchor)
  - Revocation Information for the signing 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)

It results in the following set of POEs:

Content	Exists at time
The signature	T5
The signing certificate (and other certificates required to form a chain to a trust anchor)	T5
Revocation Information for the signing 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

- Step 4c: the time-stamp validation process is performed (clause 5.4):
  - The Basic Signature validation of the signature on the time-stamp token 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*, the past signature validation process for the time-stamp is performed (see clause 5.6.2.4):
  - The past certificate validation for the TSA certificate is performed:
    - The prospective chain can be built (all information is present 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 succeeds.
    - The validation -time sliding process is performed with the following inputs: the prospective chain and the set of POEs.
      - Control-time is current time.
      - Revocation objects for the TSA-certificate are in the set of POE.
      - Proof of existence of the relevant objects exists at T5.
      - The revocation object is assumed not to be fresh and thus the control-time is set to the time this revocation object has been created (T7).
      - The certificate constraints and cryptographic constraints are applied to the chain, and succeed by assumption.
      - *PASSED* and control-time T7 are returned.
    - Since the current time status is *INDETERMINATE/OUT_OF_BOUNDS_NO_PO*E and there is a POE for the signature time-stamp at T5 before T7, the past signature validation returns *PASSED*.
- The POE-extraction process is performed for that time-stamp and a new list of POEs is generated.

Content	Exists at time
The signature	T3
The signing certificate (and other certificates required to form a chain to a trust anchor)	T3
Revocation Information for the signing 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

- The past signature validation process for the signature is performed:
  - The past certificate validation is performed for the signing certificate:
    - Certificate chain can be built by assumption.
    - Certificate path validation succeeds.
    - The validation time sliding process is performed for the signing certificate:
      - Control-time is current time.
      - A POE exists at the current time for the CA certificate as well as the corresponding revocation info status.
      - Since the CA is revoked at t8, control-time takes this value (assuming that freshness does not apply).
      - Proof of existence of the relevant objects for the signing certificate exists at T3 before T8.
      - The revocation object is assumed to be fresh and thus the change control-time is unchanged.
      - The certificate constraints and cryptographic constraints are applies to the chain, and succeed by assumption.
      - *PASSED* and control-time T8 are returned.
    - Since the current time status is *INDETERMINATE/REVOKED_CA_NO_POE* and there is a POE for the signature at T3 before T8, the past signature validation returns *PASSED.*
- The validation algorithm returns a final *TOTAL- PASSED* plus the validation report.

Annex B (informative): Signature Classes and AdES Signatures

This annex maps the signature classes specified in the present document with signature levels specified in the specification of AdES-Formats (ETSI TS 119 122-1 [i.2], ETSI TS 119 122-2 [i.3], ETSI TS 119 132-1 [i.4], ETSI TS 119 132-2 [i.5], ETSI TS 119 142-1 [i.6] and ETSI TS 119 142-2 [i.7]).

AdES-Level		Basic Signature	Signature With Time	signatures with Long-Term Validation Data	signatures with Archival Data
Baseline	**Extended**				
CAdES-B-B, XAdES-B-B, B-level PAdES	CAdES-BES, CAdES-EPES XAdES-BES, XAdES-EPES PAdES-BES, PAdES-EPES	✖			
CAdES-B-T, XAdES-B-T, T-level PAdES	CAdES-E-T, CAdES-E-C, CAdES-E-X, XAdES-E-T, XAdES-E-C, XAdES-E-X		✖		
CAdES-B-LT, XAdES-B-LT LT-level PAdES	CAdES-E-X-L XAdES-E-X-L			✖	
CAdES-B-LTA, XAdES-B-LTA LTA-level PAdES	CAdES-E-A XAdES-E-A PAdES-LTV				✖

History

Document history		
V1.0.0	July 2015	ETSI EN 319 102-1 Approval Procedure AP [#]: [START] to [END]
V1.0.1	July 2015	Publication (same technical content as ETSI EN 319 102-1 V1.0.0)