# ETSI TS 102 853 V1.1.2 (2012-10)

Technical Specification

**Electronic Signatures and Infrastructures (ESI); Signature validation procedures and policies**

Reference

RTS/ESI-000074rev

Keywords

electronic signature, security, trust services

# Contents

## Intellectual Property Rights

## Foreword

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

## Introduction

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special consideration on signature validation of "old" electronic signatures, where certificates may have expired or been revoked or even the usage period of algorithms have been exceeded.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

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

Signature validation is driven by a signature validation policy. The algorithm presented here supports such policies. It is assumed that the validator, represented by the driving application, provides such a policy in possibly different forms - as a formal policy, as a set of configuration parameters, or by the way the algorithm has been implemented.

To avoid confusing terms, the term *constraint* is used for a single policy rule that influences decisions made by the algorithm. A formal signature policy, as specified in [i.3], can provide a set of constraints, which may be used exclusively or may be combined with other constraints (e.g. coming from local configuration).

# 1 Scope

The present document specifies procedures for establishing whether an electronic signature is technically valid based on the considerations specified in the present document and the validation constraints are applied to the verification procedures. These constraints may be specified as part of a formal signature policy.

It is outside the scope of the present document as to whether a signature is accepted by the relying party and specifically if it bears legal validity.

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

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

# 2 References

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

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

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

## 2.1 Normative references

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

[1] ETSI TS 101 903 (V1.4.2): "Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES)".

[2] ETSI TS 101 733 (V2.1.1): "Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES)".

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

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

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

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

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

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

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

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

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

[12] ETSI TS 102 778-1: "Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 1: PAdES Overview - a framework document for PAdES".

[13] ETSI TS 102 778-3: "Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 3: PAdES Enhanced - PAdES-BES and PAdES-EPES Profiles".

[14] ETSI TS 102 778-4: "Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 4: PAdES Long Term - PAdES LTV Profile".

[15] ETSI TS 102 778-5: "Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 5: PAdES for XML Content - Profiles for XAdES signatures".

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

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

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

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

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

## 2.2 Informative references

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

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

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

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

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

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

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

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

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

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

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

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

# 3 Definitions and abbreviations

## 3.1 Definitions

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

**Advanced Electronic Signature (AdES):** advanced electronic signature means an electronic signature that meets the following requirements [9]:

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

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

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

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

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

**data to be signed:** data (e.g. a document or parts of a document) to be signed as well as any signature attributes that are bound together with the data by the signature

NOTE: Data To Be Signed is the input to the cryptographic signing algorithm. The specific way that Data to be Signed and any signature attributes are fed as input is defined in the specification for the signature type used.

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

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

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

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

**signature policy:** set of rules for the creation and validation of an electronic signature, under which the signature can be determined to be valid in a particular transactions context

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

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

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

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

**signature validation policy:** set of rules (constraints) that specify how to validate the signature

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

**signed data object (s):** document(s) or parts of the document(s) for which an electronic signature has been generated, along with the electronic signature

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

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

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

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

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

## 3.2 Abbreviations

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

| | |
|---|---|
| AdES | Advanced Electronic Signature |
| BES | Basic Electronic Signature |
| CA | Certification Authority |
| CAdES | CMS Advanced Electronic Signatures |
| CD | Commission Decision |
| CRL | Certificate Revocation List |
| CV | Cryptographic Verification |
| DA | Driving Application |
| DN | Distinguished Name |
| EC | European Commission |
| EPES | Explicit Policy-based Electronic Signature |
| ERS | Evidence Record Syntax |
| IP | Internet Protcol |
| ISC | Identification of the Signer's Certificate |
| LCP | Lightweight Certificate Policy |
| LDAP | Lightweight Directory Access Protocol |
| LT | Long Term |
| LTA | Long-Term with Archive Time Stamp |
| LTV | Long Term Validation |
| NCP | Normalized Certificate Policy |
| NO_POE | NO Proof Of Existence |
| OCSP | Online Certificate Status Provider |
| OID | Object Identifier |
| PAdES | PDF Advanced Electronic Signatures |
| PKIX | Public Key Infrastructure X. 509 |
| POE | Proof Of Existence |
| QCP | Qualified Certificate Policy |
| RFC | Request For Comment |
| RSA | Rivest-Shamir-Adleman |
| SAV | Signature Acceptance Validation |
| SSCD | Secure Signature Creation Device |
| ST | Short-Term |
| SVA | Signature Validation Application |
| TA | Trust Anchor |
| TSA | Time Stamping Authority |
| TSL | Trust-service Status List |
| TST | Time-Stamp Token |
| URI | Uniform Resource Identifier |
| VCI | Validation Context Initialisation |
| XAdES | XML Advanced Electronic Signatures |
| XCV | X.509 Certificate Validation |
| XL | Extended Long electronic signature |
| XML | Extendable Mark-up Language |

# 4 Introduction to signature validation

A signature validation application (SVA) validates an electronic signature against a set of validation constraints and outputs a validation report. This report consists of a status indication accompanied by additional data items, providing the details of the technical validation of each of the applicable constraints. The report **may** include additional information (e.g. explanations and other information to be displayed) that has been found relevant by the SVA and may be relevant for the driving application (DA) in interpreting the results. The output of the SVA is meant to be processed by the DA (e.g. to be displayed to the verifier).

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

Checking that the signature to validate is conformant to the applicable format (e.g. CMS/CAdES, XML-DSig/XAdES, etc.) shall be done by the SVA prior to any subsequent processing. In case the signature is not conformant to the required format, the SVA shall fail with *INVALID/FORMAT_FAILURE* together with details about the format error(s). While some of these format checks are detailed in the present document (when they are relevant to a specific validation step), format checking is out of the scope of the present document. These checks include checking that the syntax of the signature is conformant to the appropriate specification but also any additional checking mandated by that specification for specific signature attributes (e.g. checking that what is time-stamped by a time-stamp token in the signature is really what shall be time-stamped according to the appropriate specification).

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

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

The present document presents the validation process in the form of algorithms to be implemented by a conforming signature validation application. Conforming implementations however are not required to implement these algorithms but shall provide behaviour that is functionally equivalent.

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

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

## 4.1 Status indication of the signature validation process

With regards to the validation report, Table 1 lists the possible values of the main status indication and their semantics. The DA can present the report in a way meaningful to the verifier. In all cases, the signature validation process shall output an indication of the policy or set of constraints against which the signature has been validated and **may** output additional data items extracted from the signature.

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

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

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

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

Indications returned by SVAs shall conform to the following rules:

- **When the result is due to be VALID or INVALID:**
    a) Any execution of a conformant SVA with the same inputs will return VALID or INVALID, respectively.
    b) Any execution of a conformant SVA with the same inputs + additional validation data (e.g. more certificates) will return the same result as it has returned in a) (i.e. VALID or INVALID).
- **When the result is due to be INDETERMINATE:**
    a) Any execution of a conformant SVA with the same inputs will return INDETERMINATE.
    b) Any execution of a conformant SVA with the same inputs + additional validation data will return VALID, INVALID or INDETERMINATE.

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

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

NOTE 4: The status indicators VALID, INVALID and INDETERMINATE are also used in the building blocks specified in the following clauses. For the building-blocks, these statuses only represent the result of the operation performed in the block and not necessarily the result of the overall signature validation. Any sub-indicators used in the building blocks have the semantics of the sub-indicators in Table 2.

**Table 1: Status indications of the signature validation process**

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

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

**Table 2: Validation Report Structure**

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

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

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

## 4.2 Validation Constraints

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

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

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

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

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

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

## 4.3 X.509 certificate meta-data

X.509 certificate meta-data is additional information that is associated to a given certificate, a CRL or an OCSP.

NOTE: Such meta-data may be required to allow the SVA to correctly validate a signature, if the SVA is likely unable to find this information by itself or in case there may be conflicting information the SVA would not be unable to resolve on its own. For example: If the validation policy requires a qualified certificate, but this information is not contained in the certificate itself, but the certificate is known to be a qualified one, the DA can make this information available to the SVA as meta-data.

Certificate meta-data may, e.g. be:

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

## 4.4 Trust Management

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

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

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

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

## 4.5 The concept of revocation freshness

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

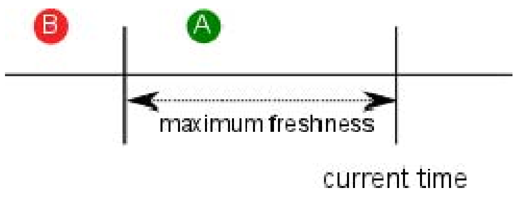

**Figure 1: Freshness**

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

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

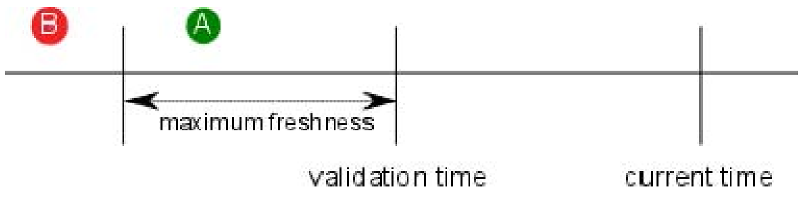

**Figure 2: Freshness in the past**

## 5 Basic Building Blocks

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

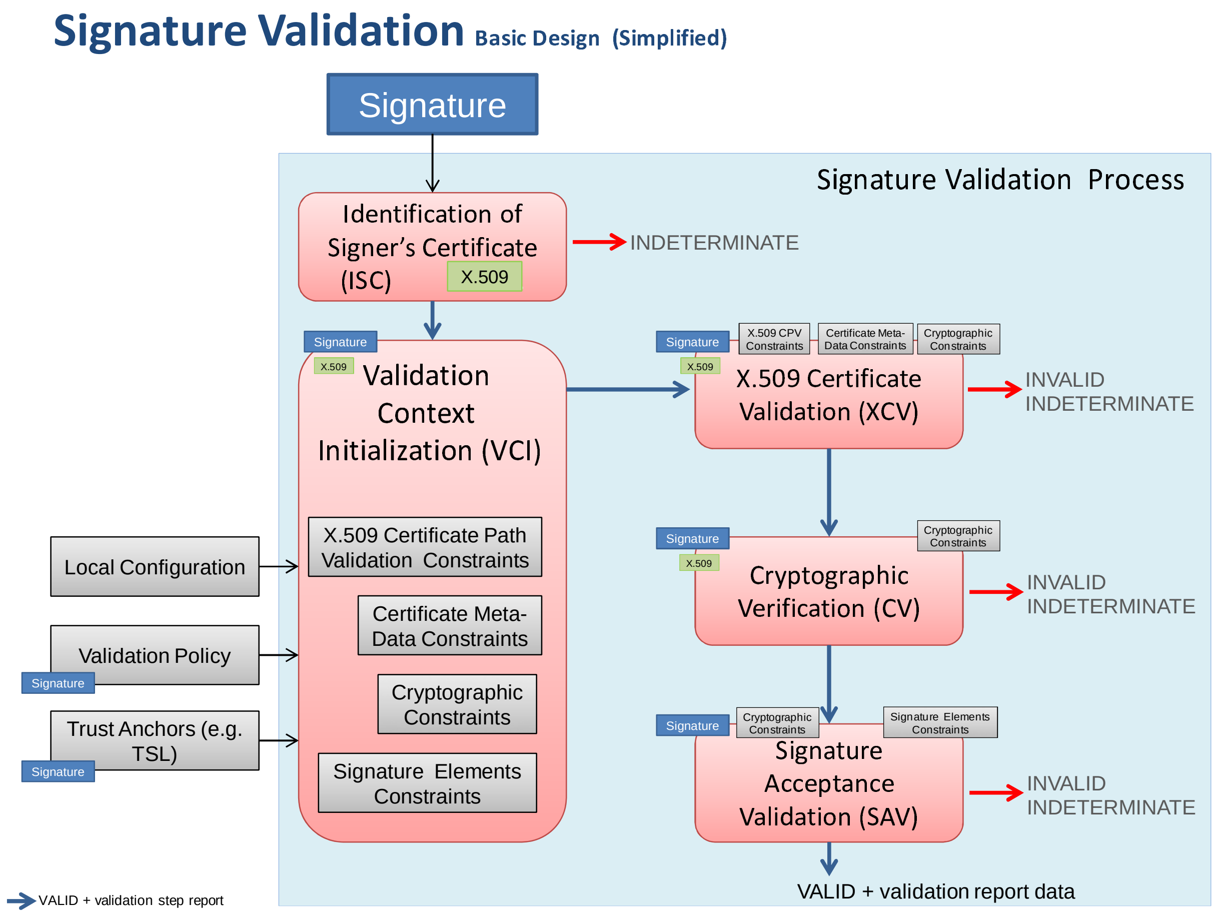
Signature Validation Basic Design (Simplified)
Signature
Signature Validation Process
Identification of Signer's Certificate (ISC)
X.509
INDETERMINATE
Signature
X.509
Validation Context Initialization (VCI)
X.509 Certificate Path Validation Constraints
Certificate Meta-Data Constraints
Cryptographic Constraints
Signature Elements Constraints
Local Configuration
Validation Policy
Signature
Trust Anchors (e.g. TSL)
Signature
X.509 CPV Constraints
Certificate Meta-Data Constraints
Cryptographic Constraints
Signature
X.509
X.509 Certificate Validation (XCV)
INVALID
INDETERMINATE
Cryptographic Constraints
Signature
X.509
Cryptographic Verification (CV)
INVALID
INDETERMINATE
Signature
Cryptographic Constraints
Signature Elements Constraints
Signature Acceptance Validation (SAV)
INVALID
INDETERMINATE
VALID + validation report data
VALID + validation step report

**Figure 3: Signature Validation**

## 5.1 Identification of the Signer's Certificate (ISC)

### 5.1.1 Description

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

### 5.1.2 Inputs

**Table 3: Inputs to the ISC process**

| Input | Requirement |
|---|---|
| Signature | Mandatory |
| Signer's Certificate | Optional |

### 5.1.3 Outputs

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

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

### 5.1.4 Processing

The common way to unambiguously identify the signer's certificate is by using a property/attribute of the signature containing a reference to it, which includes the digest computed over the certificates encoded value. The certificate or a reference to the certificate can either be found in the signature or it can be obtained using external sources. The signer's certificate may also be provided by the DA. If the certificate cannot be retrieved, the indication *INDETERMINATE* will be the result.

Clauses 5.1.4.1 to 5.1.4.3 provide specific processing details for each AdES signature type (i.e. XAdES, CAdES or PAdES), once the certificate has been retrieved.

#### 5.1.4.1 XAdES processing

The signing certificate shall be checked against all references present in the `ds:SigningCertificate` property, if present, since one of these references shall be a reference to the signing certificate [1]. The following steps shall be performed:

1) Take the first child of the property and check that the content of `ds:DigestValue` matches the result of digesting the signing certificate with the algorithm indicated in `ds:DigestMethod`. If they do not match, take the next child and repeat this step until a matching child element has been found or all children of the element have been checked. If they do match, continue with step 2. If the last element is reached without finding any match, the validation of this property shall be taken as failed and *INVALID/FORMAT_FAILURE* is returned.
2) If the `ds:KeyInfo` contains the `ds:X509IssuerSerial` element, check that the issuer and the serial number indicated in that element and `IssuerSerial` from `SigningCertificate` are the same. If they do not match, the validation of this property shall be taken as failed and *INDETERMINATE* is returned.

#### 5.1.4.2 CAdES processing

The signing certificate shall be checked against the references present in one of the following attributes: ESS-signing-certificate, ESS-signing-certificate-v2 or Other-signing-certificate, since one of these attributes shall contain a reference to the signing certificate. For doing this, the following tasks shall be performed:

1) Take the first element of the attribute and check that the content of the field containing the digest value matches the result of digesting the signing certificate with the algorithm implicitly or explicitly indicated in the reference attribute. If they match, continue with step 2. Otherwise the validation of this attribute shall be taken as failed and *INVALID/FORMAT_FAILURE* is returned.
2) Compare the details of the issuer's name and the serial number of the certificate with those indicated in the reference. If any of them does not match, the validation of this attribute shall be taken as failed and *INDETERMINATE* is returned.

#### 5.1.4.3 PAdES processing

The signing certificate shall be checked against the references present in one of the following attributes: ESS-signing-certificate or ESS-signing-certificate-v2, since one of these attributes shall contain a reference to the signing certificate. For doing this, follow the same steps as for CAdES (see clause 5.1.4.2).

## 5.2 Validation Context Initialization (VCI)

### 5.2.1 Description

This process consists in initializing the validation constraints (chain constraints, cryptographic constraints, signature constraints) and parameters (X.509 validation parameters, certificate meta-data) that will be used to validate the signature. The constraints and parameters may be initialized from any of the sources listed in clauses 4.2, 4.3 and 4.4.

### 5.2.2 Inputs

**Table 4: Inputs to the VCI process**

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

### 5.2.3 Outputs

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

In case of success, the process outputs the following:

**Table 5: Output of the VCI process**

| Output |
|---|
| X.509 Validation Parameters |
| Certificate meta-data |
| Chain Constraints |
| Cryptographic Constraints |
| Signature Constraints |

### 5.2.4 Processing

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

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

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

#### 5.2.4.1 Processing commitment type indication

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

###### 5.2.4.1.1 XAdES Processing

If the signature is a XAdES signature, the SVA shall check that each `xades:ObjectReference` element within the `xades:CommitmentTypeIndication` actually references a `ds:Reference` element present in the signature. If any of these elements does not refer to one of the `ds:Reference` elements, then the SVA shall assume that a format failure has occurred during the verification and return *INVALID/FORMAT_FAILURE* with an indication that the validation failed to an invalid commitment type property.

#### 5.2.4.2 Processing Signature Policy Identifier

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

1) Retrieve the electronic document containing the details of the policy, and identified by the contents of the property/attribute.

2) If the signature is a XAdES signature, apply the transformations indicated in the `ds:Transforms` element of `xades:SignaturePolicyId` element. If the signature is not a XAdES signature, go to step 3.

3) Obtain the digest of the resulting document against which the digest value present in the property/attribute will be checked:

    a) If the resulting document is based on TR 102 272 [i.2], use the digest value present in the `SignPolicyDigest` element from the resulting document. Check that the digest algorithm indicated in the `SignPolicyDigestAlg` from the resulting document is equal to the digest algorithm indicated in the property.

    b) If the resulting document is based on TR 102 038 [i.3], use the digest value present in `signPolicyHash` element from the resulting document. Check that the digest algorithm indicated in the `signPolicyHashAlg` from the resulting document is equal to the digest algorithm indicated in the attribute.

    c) In all other cases, compute the digest using the digesting algorithm indicated in the children of the property/attribute.

4) Check that the digest obtained in the previous step is equal to the digest value indicated in the children of the property/attribute.

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

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

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

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

## 5.3 X.509 Certificate Validation (XCV)

### 5.3.1 Description

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

### 5.3.2 Inputs

**Table 6: Inputs to the XCV process**

| Input | Requirement |
|---|---|
| Signature | Mandatory |
| Signer's certificate | Mandatory |
| X.509 Validation Parameters | Mandatory |
| Certificate meta-data | Optional |
| Chain Constraints | Optional |
| Cryptographic Constraints | Optional |

### 5.3.3 Outputs

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

**Table 7: Output of the XCV process**

| Indication | |
|---|---|
| *VALID* | |
| *INDETERMINATE* | *NO_CERTIFICATE_CHAIN_FOUND* |
| | *OUT_OF_BOUNDS_NO_POE* |
| | *REVOKED_NO_POE* |
| | *CRYPTO_CONSTRAINTS_FAILURE_NO_POE* |
| *INVALID* | *CHAIN_CONSTRAINTS_FAILURE* |

### 5.3.4 Processing

This process consists of the following steps:

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

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

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

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

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

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

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

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

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

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

f) If the certificate path validation returns a failure indication with any other reason, go to step 2.

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

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

6) Return the chain with the indication VALID.

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

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

## 5.4 Cryptographic Verification (CV)

### 5.4.1 Description

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

### 5.4.2 Inputs

**Table 8: Inputs to the CV process**

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

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

### 5.4.3 Outputs

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

**Table 9: Outputs of the CV process**

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

### 5.4.4 Processing

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

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

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

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

## 5.5 Signature Acceptance Validation (SAV)

### 5.5.1 Description

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

### 5.5.2 Inputs

**Table 10: Inputs to the SVA process**

| Input | Requirement |
|---|---|
| Signature | Mandatory |
| Cryptographic verification output | Optional |
| Cryptographic Constraints | Optional |
| Signature Constraints | Optional |

### 5.5.3 Outputs

The process outputs one of the following indications:

**Table 11: Outputs of the SVA process**

| Indication | | Description | Additional data items |
|---|---|---|---|
| *VALID* | | The signature is conformant with the validation constraints. | |
| *INVALID* | *SIG_CONSTRAINTS_FAILURE* | The signature is not conformant with the validation constraints. | The process shall output: <br>• The set of constraints that are not verified by the signature. |
| *INDETERMINATE* | *CRYPTO_CONSTRAINTS_FAILURE_NO_POE* | At least one of the algorithms used in validation of the signature together with the size of the key, if applicable, used with that algorithm is no longer considered reliable. | The process shall output: <br>• 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. |

### 5.5.4 Processing

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

- If the constraint necessitates processing a property/attribute in the signature, perform the processing of the property/attribute as specified from clauses 5.5.4.1 to 5.5.4.8.
- If at least one of the algorithms that have been used in validation of the signature or the size of the keys used with such an algorithm is no longer considered reliable, return *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE* together with the list of algorithms and key sizes, if applicable, that are concerned and the time for each of the algorithms up to which the respective algorithm was considered secure.

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

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

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

#### 5.5.4.1 Processing AdES properties/attributes

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

#### 5.5.4.2 Processing signing certificate reference constraint

If the `SigningCertificate` property contains references to other certificates in the path, the verifier shall check each of the certificates in the certification path against these references as specified in steps 1 and 2 in clause 5.1.4.1 (respectively clause 5.1.4.2) for XAdES (respectively CAdES).

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

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

#### 5.5.4.3 Processing claimed signing time

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

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

#### 5.5.4.4 Processing signed data object format

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

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

#### 5.5.4.5 Processing indication of production place of the signature

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

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

#### 5.5.4.6 Processing Time-stamps on signed data objects

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

1) Perform the Validation Process for AdES Time-Stamps as defined in clause 7 with the time-stamp token of the content-time-stamp attribute.

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

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

#### 5.5.4.7 Processing Countersignatures

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

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

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

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

#### 5.5.4.8 Processing signer attributes/roles

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

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

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

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

# 6 Basic Validation Process

## 6.1 Description

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

## 6.2 Inputs

**Table 12: Inputs to BES/EPES validation**

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

## 6.3 Outputs

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

## 6.4 Processing

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

The following steps shall be performed:

1) Identify the signer's certificate: Perform the Signer's Certificate Identification process (see clause 5.1) with the signature and the signer's certificate, if provided as a parameter. If it returns *INDETERMINATE*, terminate with *INDETERMINATE* and associated information, otherwise go to the next step.

2) Initialize the validation constraints and parameters: Perform the Validation Context Initialization process (see clause 5.2).

3) Validate the signer's certificate: Perform the X.509 Certificate Validation process (see clause 5.3) with the following inputs:

   a) The signature.

   b) The signer's certificate obtained in step 1.

   c) X.509 Validation Parameters, Certificate meta-data, Chain Constraints and Cryptographic Constraints obtained in step 2:

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

      - If the process returns *INDETERMINATE/REVOKED_NO_POE*: If the signature contains a content-time-stamp attribute, perform the Validation Process for AdES Time-Stamps as defined in clause 7. If it returns *VALID* and the generation time of the time-stamp token is after the revocation time, terminate with *INVALID/REVOKED*. In all other cases, terminate with *INDETERMINATE/REVOKED_NO_POE*.

      - If the process returns *INDETERMINATE/OUT_OF_BOUNDS_NO_POE*: If the signature contains a content-time-stamp attribute, perform the Validation Process for AdES Time-Stamps as defined in clause 7. If it returns *VALID* and the generation time of the time-stamp token is after the expiration date of the signer's certificate, terminate with *INVALID/EXPIRED*. In all other cases, terminate with *INDETERMINATE/OUT_OF_BOUNDS_NO_POE*.

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

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

   a) The signature.

   b) The certificate chain returned in the previous step.

   c) The signed data object(s).

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

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

   a) The signature.

   b) The Cryptographic Constraints.

   c) The Signature Constraints.

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

      ▪ If the process returns INDETERMINATE*/CRYPTO_CONSTRAINTS_FAILURE_NO_POE* and the material concerned by this failure is the signature value: If the signature contains a content-time-stamp attribute, perform the Validation Process for AdES Time-Stamps as defined in clause 7. If it returns *VALID* and the algorithm(s) concerned were no longer considered reliable at the generation time of the time-stamp token, terminate with *INVALID/CRYPTO_CONSTRAINTS_FAILURE*. In all other cases, terminate with *INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE*.

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

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

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

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

# 7 Validation Process for Time-Stamps

## 7.1 Description

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

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

## 7.2 Inputs

**Table 13: Inputs to time stamp validation**

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

## 7.3 Outputs

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

## 7.4 Processing

The following steps shall be performed:

1) Token signature validation: perform the validation process for BES signature (see clause 6) with the time-stamp token. In all the steps of this process, take into account that the signature to validate is a time-stamp token (e.g. to select TSA trust-anchors). If this step ends with a success indication, go to the next step. Otherwise, fail with the indication and information retuned by the validation process.

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

# 8 Validation Process for AdES-T

## 8.1 Description

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

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

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

## 8.2 Inputs

**Table 14: Inputs to AdES-T validation**

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

## 8.3 Outputs

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

## 8.4 Processing

The following steps shall be performed:

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

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

2) Signature validation: Perform the validation process for BES signatures (see clause 6) with all the inputs, including the processing of any signed attributes/properties as specified. If this validation outputs *VALID, INDETERMINATE/CRYPTO_CONSTRAINTS_FAILURE_NO_POE, INDETERMINATE/REVOKED_NO_POE or INDETERMINATE/OUT_OF_BOUNDS_NO_POE*, go to the next step. Otherwise, terminate with the returned status and information.

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

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

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

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

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

   a) Message imprint verification: For each time-stamp token in the set of signature time-stamp tokens, do the message imprint verification as specified in clauses 8.4.1 or 8.4.2 depending on the type of the signature. If the verification fails, remove the token from the set.

   b) Time-stamp token validation: For each time-stamp token remaining in the set of signature time-stamp tokens, the SVA shall perform the time-stamp validation process (see clause 7):

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

      ▪ In all remaining cases, remove the time-stamp token from the set of signature time-stamp tokens and try the next token.

5) Comparing times:

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

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

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

d) For each time-stamp token remaining in the set of signature time-stamp tokens, check the coherence in the values of the times indicated in the time-stamp tokens. They shall be posterior to the times indicated in any time-stamp token computed on the signed data (i.e. any `content-time-stamp` signed attributes in CAdES or any `AllDataObjectsTimeStamp` or `IndividualDataObjectsTimeStamp` signed present properties in XAdES). The SVA shall apply the rules specified in RFC 3161 [11], clause 2.4.2 regarding the order of time-stamp tokens generated by the same or different TSAs given the `accuracy` and `ordering` fields' values of the `TSTInfo` field, unless stated differently by the Signature Constraints. If all the checks end successfully, go to the next step. Otherwise return *INVALID/TIMESTAMP_ORDER_FAILURE*.

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

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

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

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

NOTE 5: In the algorithm above, the signature-time-stamp protects the signature against the revocation of the signer's certificate (step 5-a) but not against expiration. The latter case requires validating the signer's certificate in the past (see clause 9).

### 8.4.1 Message Imprint Verification of the signature-timestamp for XAdES

1) The SVA shall take the `ds:SignatureValue` element and canonicalize it using the algorithm indicated in `CanonicalizationMethod` element of the property, if present. Otherwise use the standard canonicalization method as specified by XMLDSIG [10].

2) The SVA shall compute the digest of the resulting byte stream using the algorithm indicated in the time-stamp token and shall check if this value matches the values present in that token.

### 8.4.2 Message Imprint Verification of the signature-time-stamp for CAdES/PAdES

1) The SVA shall take the `signature` field of the CAdES signature, encode it and compute the digest of the resulting byte stream using the algorithm indicated in the time-stamp token.

2) The SVA shall check if the value obtained in the first step is the same as the digest present in the time-stamp token.

# 9 Validation of LTV forms

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

In particular, this is useful in the case where the SVA shall take as input, in addition to the basic signature to validate, additional evidences derived from previous validation (e.g. a proof of existence derived from the validation of a time-stamp token). The process is built on the building block described in clause 5 and the additional building blocks defined in clause 9.3.

The algorithms in this clause use attribute terminology used in the CAdES and XAdES specifications. The same algorithms apply to PAdES signatures by considering the equivalent structures defined in PAdES.

## 9.1 The concept of Proof Of Existence (POE)

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

## 9.2 Additional Building blocks

### 9.2.1 Past certificate validation

#### 9.2.1.1 Description

This process validates a certificate at a date/time which may be in the past. This may become necessary in the LTV settings when a compromising event (for instance, the end-entity certificate expires) prevents the traditional certificate validation algorithm (see clause 5.3) to asserting the validation status of a certificate (for instance, in case the end-entity certificate is expired at the current time, the traditional validation algorithm will return *INDETERMINATE/OUT_OF_BOUNDS_NO_POE* due to the step 1).

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

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

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

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

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

It is important to note that when all the certificates in the chain can be validated at the current time, the control-time never slides and the algorithm boils down to the traditional certificate validation algorithm described in clause 5.3.

The process below builds a prospective certificate chain in a very same way as in clause 5.3 except that the X.509 validation algorithm is performed at a determined date in the past (instead of the current date/time) and without any revocation checking. For each such chain, the sliding algorithm is executed to calculate the control-time.

#### 9.2.1.2 Input

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

#### 9.2.1.3 Output

| Indication | |
|---|---|
| *VALID* | |
| *INDETERMINATE* | *CHAIN_CONSTRAINTS_FAILURE* |
| | *NO_CERTIFICATE_CHAIN_FOUND* |
| | *NO_POE* |

#### 9.2.1.4 Processing

The following steps shall be performed:

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

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

   b) Otherwise, go to the next step.

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

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

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

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

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

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

### 9.2.2 Control-time sliding process

#### 9.2.2.1 Description

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

#### 9.2.2.2 Input

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

#### 9.2.2.3 Output

| Indication | |
|---|---|
| *VALID* | |
| *INDETERMINATE* | *NO_POE* |

#### 9.2.2.4 Processing

The following steps shall be performed:

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

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

    a) Find revocation status information satisfying the following:

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

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

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

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

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

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

        ▪ If the certificate is not marked as revoked.

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

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

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

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

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

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

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

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

## 9.2.3 POE extraction

### 9.2.3.1 Description

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

- The time-stamp has been accepted as *VALID*.
- The cryptographic hash function used in the time-stamp (*MessageImprint.hashAlgorithm*) is considered reliable at the generation time of the time-stamp.

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

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

- If we have a POE for h(d) at a date $T_1$,where h is a cryptographic hash function and d is some data (e.g. a certificate).
- And h is asserted in the cryptographic constraints to be trusted until at least a date T after $T_{1.}$
- And we have a POE for d at a date T after $T_{1.}$

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

### 9.2.3.2 Input

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

### 9.2.3.3 Output

A set of POEs.

### 9.2.3.4 Processing

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

##### 9.2.3.4.1 Extraction from a time-stamp on the signature

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

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

##### 9.2.3.4.2 Extraction from a time-stamp on certificates and revocation references

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

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

1) Add a POE for the hash value h(C) of the certificate C (respectively h(R) of the revocation status information R).

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

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

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

1) Do the extraction process from a time-stamp on the signature (see clause 9.2.3.4.1).

2) Do the extraction process from a time-stamp on certificates and revocation references (see clause 9.2.3.4.2).

##### 9.2.3.4.4 Extraction from an archive-time-stamp

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

1) Add a POE for each signed object.

2) Add a POE for the signature value.

3) Add a POE for each certificate and revocation status information present in the signature.

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

##### 9.2.3.4.5 Extraction from a long-term-validation attribute

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

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

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

##### 9.2.3.4.6 Extraction from a PDF document time-stamp

This process applies only to PAdES [14].

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

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

## 9.2.4 Past signature validation process

### 9.2.4.1 Description

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

### 9.2.4.2 Input

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

### 9.2.4.3 Output

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

#### 9.2.4.4 Processing

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

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

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

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

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

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

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

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

## 9.3 Long Term Validation Process

### 9.3.1 Description

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

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

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

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

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

### 9.3.2 Input

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

### 9.3.3 Output

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

### 9.3.4 Processing

The following steps shall be performed:

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

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

2) Basic signature validation: Perform the validation process for AdES-T signatures (see clause 8) with all the inputs, including the processing of any signed attributes/properties as specified.

   - If the validation outputs *VALID*

     ▪ If there is no validation constraint mandating the validation of the LTV attributes/properties, go to step 9.

     ▪ Otherwise, go to step 3.

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

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

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

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

3) If there is at least one long-term-validation attribute with a poeValue, process them, starting from the last (the newest) one as follows: Perform the time-stamp validation process (see clause 7) for the time-stamp in the poeValue:

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

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

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

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

4) If there is at least one archive-time-stamp attribute, process them, starting from the last (the newest) one, as follows: perform the time-stamp validation process (see clause 7):

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

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

      ▪ If no specific constraints mandating the validity of the attribute are specified in the validation constraints, ignore the attribute and consider the next archive-time-stamp attribute.

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

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

5) If there is at least one time-stamp attribute on the references, process them, starting from the last one (the newest), as follows: perform the time-stamp validation process (see clause 7):

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

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

      ▪ If it returns VALID and the cryptographic hash function used in the time-stamp is considered reliable at the generation time of the time-stamp, perform the POE extraction process and add the returned POEs to the set of POEs. In all other cases:

      ▪ If no specific constraints mandating the validity of the attribute are specified in the validation constraints, ignore the attribute and consider the next archive-time-stamp attribute.

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

6) If there is at least one time-stamp attribute on the references and the signature value, process them, starting from the last one, as follows: perform the time-stamp validation process (see clause 7):

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

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

▪ If no specific constraints mandating the validity of the attribute are specified in the validation constraints, ignore the attribute and consider the next archive-time-stamp attribute.

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

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

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

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

▪ If no specific constraints mandating the validity of the attribute are specified in the validation constraints, ignore the attribute and consider the next archive-time-stamp attribute.

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

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

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

# Annex A (informative): Validation Constraints

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

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

## A.1 X.509 Certificate path validation constraints

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

**Table A.1**

| Constraint | Description | Reference |
|---|---|---|
| A set of trust anchor information | The DA provides the SVA a list of acceptable trust anchors as a constraint for the validation process. Such TAs are recommended to be provided in the form of (self-signed) certificates and a time until when these trust anchors were considered reliable. The TA information may be taken from:<br>• Trust points specified in signature validation policies<br>• Sets of trusted CAs, e.g. represented by their root certificates stored in the environment (like certificate trust store or list)<br>• Trust Service Status Lists as defined in [3]<br>• Trusted Lists as defined in [CD]<br>The DA may also provide the TA information to the SVA in one of these forms, if applicable. | [4], [i.1], CD 2009/767/EC [i.6] amended by CD 2010/425/EU<br>[i.3], [i.2] |
| A certification path | This constraint consists in the provision of a certification path of length 'n' from the TA down to the certificate used in creating a signed object (e.g. the signer's certificate or a time stamping certificate). The given certification path has to be used by the SVA for validation of the signature. | [4] |
| `user-initial-policy-set` | *"A set of certificate policy identifiers naming the policies that are acceptable to the DA. The user-initial-policy-set contains the special value any-policy when not concerned about certificate policy".* | [4] |
| `initial-policy-mapping-inhibit` | *"Indicates if policy mapping is allowed in the certification path".* | [4] |
| `initial-explicit-policy` | *"Indicates if the path must be valid for at least one of the certificate policies in the user-initial-policy-set".* | [4] |
| `initial-any-policy-inhibit` | *"Indicates whether the anyPolicy OID should be processed if it is included in a certificate".* | [4] |
| `initial-permitted-subtrees` | *"Indicates for each name type (e.g. X.500 distinguished names, email addresses, or IP addresses) a set of subtrees within which all subject names in every certificate in the certification path MUST fall".* | [4] |
| `initial-excluded-subtrees` | *"Indicates for each name type (e.g. X.500 distinguished names, email addresses, or IP addresses) a set of subtrees within which no subject name in any certificate in the certification path may fall".* | [4] |

**Additional Chain Constraints:**

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

**Table A.2**

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

**Additional Revocation Constraints:**

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

**Table A.3**

| Constraint | Description | Reference |
|---|---|---|
| Revocation Checking Constraints | Indicates requirements for checking certificate revocation.<br>Such constraints may specify:<br>• If revocation checking is required or not<br>• If OCSP responses or CRLs have to be used<br>One possible syntax/semantic for a set of requirement values used to express such requirements is defined in TR 102 272 [i.2] and TR 102 038 [i.3]:<br><br>***"clrCheck:*** *Checks shall be made against current CRLs (or ARLs);*<br>***ocspCheck:*** *The revocation status shall be checked using OCSP RFC 2560 [i.9];*<br>***bothCheck:*** *Both OCSP and CRL checks shall be carried out;*<br>***eitherCheck:*** *Either OCSP or CRL checks shall be carried out;*<br>***noCheck:*** *No check is mandated."* | [i.2], [i.3] |
| Revocation Freshness Constraints | Used to time constraints on revocation information. The constraints may indicate the maximum accepted difference between the issuance date of the revocation status information of a certificate and the time of validation (see clause 4.5) or require the SVA to only accept revocation information issued a certain time after the signature has been created. | present document, clause 4.4 |
| Revocation Info of expired certificates | This constraint mandates the signer's certificate used in validating the signature to be issued by a certificate authority that keeps revocation notices for revoked certificates even after they have expired for a period exceeding a given lower bound (see note). | [8], [6] |
| NOTE: The Revocation Info of expired certificates-constraint may be more efficiently implementable by not including such a CA in the list of trust anchors. | | |

**Additional Time-Stamp Trust Constraints**:

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

**Table A.4**

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

## A.2 Constraints on X.509 Certificate meta-data

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

**Table A.5**

| Constraint | Description | Reference |
|---|---|---|
| `QualifiedCertificate` | Mandates the signer's certificate used in validating the signature to be a qualified certificate as defined in Directive 1999/93/EC [9]. This status can be derived from:<br>• QcCompliance extension being set in the signer's certificate in accordance with TS 101 862 [5];<br>• QCP+ or QCP certificate policy OID being indicated in the signer's certificate policies extension (i.e. 0.4.0.1456.1.1 or 0.4.0.1456.1.2);<br>• The content of a Trusted service Status List;<br>• The content of a Trusted List through information provided in the Sie field of the applicable service entry; or<br>• Static configuration that provides such information in a trusted manner. | [5], [7], CD 2009/767/EC [i.6] amended by CD 2010/425/EU<br>DTS-ESI-000099,B.3,(h) |
| `SSCD` | Mandates the end user certificate used in validating the signature to be supported by a secure signature creation device (SSCD) as defined in Directive 1999/93/EC [9].<br>This status is derived from:<br>• QcSSCD extension being set in the signer's certificate in accordance with TS 101 862 [5];<br>• QCP+ certificate policy OID being indicated in the signer's certificate policies extension (i.e. 0.4.0.1456.1.1);<br>• The content of a Trusted service Status List;<br>• The content of a Trusted List through information provided in the Sie field of the applicable service entry; or<br>• Static configuration that provides such information in a trusted manner. | [9], [7], CD 2009/767/EC [i.6] amended by CD 2010/425/EU<br>SR 001 604 [i.10], clause B.3 (n) |
| `ForLegalPerson` | Mandates the signer's certificate used in validating the signature to be issued by a certificate authority issuing certificate as having been issued to a legal person. | CD 2009/767/EC [i.6] amended by CD 2010/425/EU<br>SR 001 604 [i.10], clause B.3,(l) |

## A.3 Cryptographic Constraints

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

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

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

## A.4 Constraints on Signature Elements

**Table A.6**

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

# Annex B (informative): Certificate Meta-Data

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

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

**Table B.1**

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

# Annex C (informative): Validation Examples

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

## C.1 General remarks and assumptions

- While validating an AdDS-T signature is specified in a separate clause (see clause 8), this has been done only to keep this special case simple. It would have been perfectly possible to use the LTV/algorithm also for the T-form. In the examples we ignore this distinction and only present the logic behind the algorithm as applicable to the examples chosen.
- These examples also assume that basic checks like cryptographic or format checks succeed. We concentrate on examples showing how the fundamental properties of an AdES signature, proving the existence of certain objects at certain times, help to validate signatures from the past.
- For all validation examples, we assume to be able to identify the signer's certificate, as it is provided within the signature.
- We assume not to have any specific constraints on the validation process unless noted otherwise.
- We assume that a valid path to a trust anchor can be built for all certificates used unless noted otherwise.
- We assume only to have the signature as an input unless noted otherwise.
- We assume that the syntax/format of all elements is ok, that all required elements are there, that time stamps and signatures have been calculated over the right data and no other similar basic flaws exist, unless noted otherwise.

## C.2 Symbols

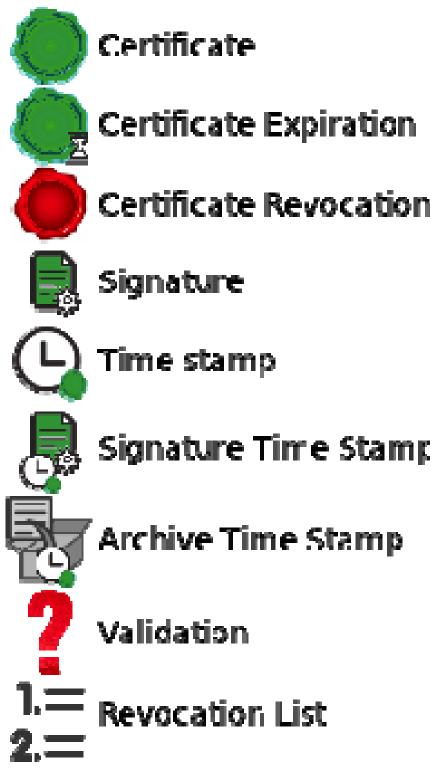

**Figure C.1: Symbols used in examples**

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

## C.3 Example 1: Revoked certificate

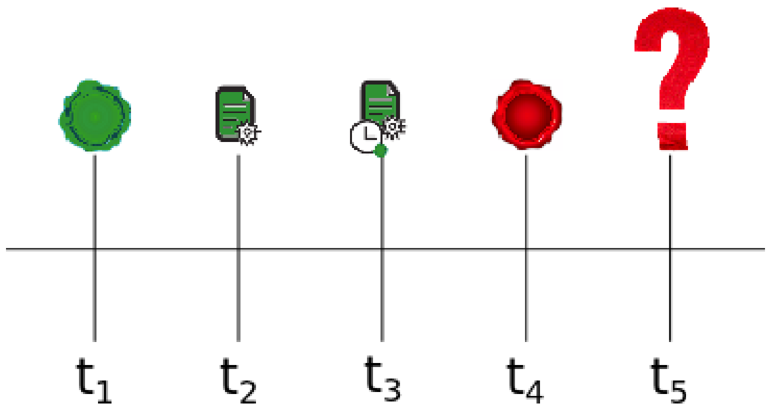

**Figure C.2: Revoked Certificate Example**

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

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

- At time t4 the certificate is revoked.
- At time t5 we try to validate the certificate.
- All other certificates that are used in the process are assumed to being still valid.

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

## C.3.1 AdES-BES/EPES

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

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

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

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

## C.3.2 AdES-T

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

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

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

- We have no constraints on time stamp delay (step 6), so we skip the next step.
- We now can return VALID and return the validation report generated to the DA (step 7).

## C.4 Example 2: Revoked CA certificate

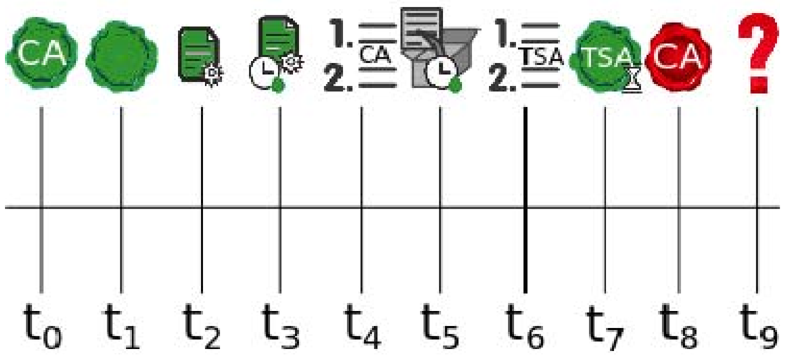

**Figure C.3: Revoked CA Certificate**

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

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

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

### C.4.1 AdES-BES/EPES

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

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

- Identify the signer's certificate: succeeds by assumption.

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

The algorithm terminates here with INDETERMINATE/REVOKED_CA*_NO_POE*, which is expected and correct.

## C.4.2 AdES-T

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

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

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

## C.4.3 LTV

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

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

We start in clause 9.2.4:

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

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

- *Perform the validation process for AdES-T signatures*: As we have seen before, this returns INDETERMINATE/REVOKED_CA*_NO_POE*, and we proceed with the algorithm, since we hope (or know) that existing time stamps may still allow us to verify the signature.

- *Archive Timestamp Validation (step 4):* We move to clause 7 for verifying the archive time stamp:
  - We perform BES-validation of the signature on the archive time stamp token, which succeeds, since we assume that the certificate of the archive-TSA has neither expired nor been revoked.
- we can extract POEs at the time of the archive timestamp (see clause 9.2.3.4.4) for:
  - The signature
  - The Signers Certificate (and other certificates required to form a chain to a trust anchor)
  - Revocation Information for the Signers Certificate (as well as for all certificates required to form a chain to a trust anchor)
  - The signature time stamp
  - The TSA Certificate related to the signature time stamp (and other certificates required to form a chain to a trust anchor)

Resulting in the following set of POEs:

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

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

        - We apply certificate constraints and cryptographic constraints to the chain, which succeed by assumption.
        - We return with VALID and control-time T7.
    - Since the current time status is INDETERMINATE/OUT_OF_BOUNDS_NO_POE and we have a POE for the signature time-stamp at T5 before T7, the past signature validation will return *VALID.*

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

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

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

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

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

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

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

- An SVA that implements the Long Term Validation Process (see clause 9.3) is willing to validate signatures conformant to any of the conformance levels (ST, T, LT and LTA).
- An SVA that implements the Validation Process for AdES-T (see clause 8) is willing to validate signatures conformant to ST, T or LT levels.
- An SVA that implements the Basic Validation Process (see clause 6) is willing to validate signatures conformant to ST level.

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

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

## History

| Document history | | |
|---|---|---|
| V1.1.1 | July 2012 | Publication |
| V1.1.2 | October 2012 | Publication |
| | | |
| | | |
| | | |