Network Working Group
Request for Comments: 3125
Category: Experimental

J. Ross
Security & Standards
D. Pinkas
Integris
N. Pope
Security & Standards

September 2001

Electronic Signature Policies

Status of this Memo

Copyright Notice

Abstract

This document defines signature policies for electronic signatures. A signature policy is a set of rules for the creation and validation of an electronic signature, under which the validity of signature can be determined. A given legal/contractual context may recognize a particular signature policy as meeting its requirements.

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

The signature policy needs to be available in human readable form so that it can be assessed to meet the requirements of the legal and contractual context in which it is being applied.

To allow for the automatic processing of an electronic signature another part of the signature policy specifies the electronic rules for the creation and validation of the electronic signature in a computer processable form. In the current document the format of the signature policy is defined using ASN.1.

The contents of this document is based on the signature policy defined in ETSI TS 101 733 V.1.2.2 (2000-12) Copyright (C). Individual copies of this ETSI deliverable can be downloaded from http://www.etsi.org.

Table of Contents

1. Introduction

This document is intended to cover signature policies which can be used with electronic signatures for various types of transactions, including business transactions (e.g., purchase requisition, contract, and invoice applications). Electronic signatures can be used for any transaction between an individual and a company, between two companies, between an individual and a governmental body, etc. This document is independent of any environment. It can be applied to any environment e.g., smart cards, GSM SIM cards, special programs for electronic signatures etc.

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document (in uppercase, as shown) are to be interpreted as described in [RFC2119].

2. Major Parties

The document uses the following terms:

* the Signature Policy Issuer;
* the Signer;
* the Verifier;
* the Arbitrator;
* Trusted Service Providers (TSP);

The Signature Policy Issuer (which is a Trusted Service Provider (TSP)) issues signatures policies that define the technical and procedural requirements for electronic signature creation, and validation/ verification, in order to meet a particular business need.

The Signer is the entity that creates the electronic signature. When the signer digitally signs over an signature policy identifier, it represents a commitment on behalf of the signing entity that the data being signed is signed under the rules defined by the signature policy.

The Verifier is the entity that validates the electronic signature, it may be a single entity or multiple entities. The verifier MUST validate the electronic signature under the rules defined by the electronic signature policy for the signature to be valid.

An arbitrator, is an entity which arbitrates disputes between a signer and a verifier. It acts as verifier when it verifies the electronic signature after it has been previously validated.

The Trusted Service Providers (TSPs) are one or more entities that help to build trust relationships between the signer and verifier. Use of TSP specific services MAY be mandated by signature policy. TSP supporting services include: user certificates, cross-certificates, time-stamping tokens,CRLs, ARLs, OCSP responses.

A Trusted Service Providers (TSPs) MAY be a Signature Policy Issuer, as Such, the TSP MUST define the technical and procedural requirements for electronic signature creation and validation, in order to meet a particular business need.

The following other TSPs are used to support the functions defined in this document:

* Certification Authorities;
* Registration Authorities;
* Repository Authorities (e.g., a Directory);
* Time-Stamping Authorities;
* One-line Certificate Status Protocol responders;
* Attribute Authorities.

Certification Authorities provide users with public key certificates.

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

Repository Authorities publish CRLs issued by CAs, , cross-certificates (i.e., CA certificates) issued by CAs, signature policies issued by Signature Policy Issuers and optionally public key certificates (i.e., leaf certificates) issued by CAs.

Time-Stamping Authorities attest that some data was formed before a given trusted time.

One-line Certificate Status Protocol responders (OSCP responders) provide information about the status (i.e., revoked, not revoked, unknown) of a particular certificate.

Attributes Authorities provide users with attributes linked to public key certificates

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

3. Signature Policy Specification

A signature policy specification includes general information about the policy, the validation policy rules and other signature policy information.

This document mandates that:

* an electronic signature must be processed by the signer and verifier in accordance with the signature policy referenced by the signer;
* the signature policy referenced by the signer must be identifiable by an Object Identifier;
* there must exist a specification of the signature policy;
* for a given signature policy there must be one definitive form of the specification which has a unique binary encoding;
* a hash of the definitive specification, using an agreed algorithm, must be provided by the signer and checked by the verifier.

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

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

This document defines an ASN.1 based syntax that may be used to define a structured signature policy. Future versions of this document may include structured a signature policy specification using XML.

3.1 Overall ASN.1 Structure

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

This ASN.1 syntax is encoded using the Distinguished Encoding Rules (DER).

In this structure the policy information is preceded by an identifier for the hashing algorithm used to protect the signature policy and followed by the hash value which must be re-calculated and checked whenever the signature policy is passed between the issuer and signer/verifier.

The hash is calculated without the outer type and length fields.

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

SignPolicyHash ::= OCTET STRING

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

SignPolicyId ::= OBJECT IDENTIFIER

PolicyIssuerName ::= GeneralNames

FieldOfApplication ::= DirectoryString
```

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

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

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

The signPolExtensions is a generic way to extend the definition of any sub-component of a signature policy.

3.2 Signature Validation Policy

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

The signature validation policy is described as follows:

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

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

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

3.3 Common Rules

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

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

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

* signerAndVeriferRules;
* signingCertTrustCondition;
* timeStampTrustCondition.

3.4 Commitment Rules

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

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

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

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

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

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

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

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

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

3.5 Signer and Verifier Rules

The following rules apply to the format of electronic signatures defined using [ES-FORMATS].

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

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

3.5.1 Signer Rules

The signer rules identify:

* if the eContent is empty and the signature is calculated using a hash of signed data external to CMS structure.

* the CMS signed attributes that must be provided by the signer under this policy;

* the CMS unsigned attribute that must be provided by the signer under this policy;

* whether the certificate identifiers from the full certification path up to the trust point must be provided by the signer in the SigningCertificate attribute;

* whether a signer’s certificate, or all certificates in the certification path to the trust point must be by the signer in the * certificates field of SignedData.

```
SignerRules ::= SEQUENCE {
        externalSignedData         BOOLEAN      OPTIONAL,
                    -- True if signed data is external to CMS structure
                         -- False if signed data part of CMS structure
                         -- Not present if either allowed
        mandatedSignedAttr          CMSAttrs,
                                  -- Mandated CMS signed attributes
        mandatedUnsignedAttr        CMSAttrs,
                                  -- Mandated CMS unsigned attributed
        mandatedCertificateRef      [0] CertRefReq DEFAULT signerOnly,
                                  -- Mandated Certificate Reference
```

```
        mandatedCertificateInfo    [1] CertInfoReq DEFAULT none,
                                   -- Mandated Certificate Info
        signPolExtensions          [2] SignPolExtensions       OPTIONAL
                                                    }

CMSattrs ::= SEQUENCE OF OBJECT IDENTIFIER
```

The mandated SignedAttr field must include the object identifier for all those signed attributes required by this document as well as additional attributes required by this policy.

The mandatedUnsignedAttr field must include the object identifier for all those unsigned attributes required by this document as well as additional attributes required by this policy. For example, if a signature time-stamp <see section 1.1) is required by the signer the object identifier for this attribute must be included.

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

```
CertRefReq ::= ENUMERATED {
                                signerOnly (1),
           -- Only reference to signer cert mandated
                                fullpath (2)

           -- References for full cert path up to a trust point required
                                        }
```

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

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

3.5.2 Verifier Rules

The verifier rules identify:

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

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

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

3.6 Certificate and Revocation Requirement

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

3.6.1 Certificate Requirements

The certificateTrustTrees identifies a set of self signed certificates for the trust points used to start (or end) certificate path processing and the initial conditions for certificate path validation as defined RFC 2459 [7] section 4. This ASN1 structure is used to define policy for validating the signing certificate, the TSA's certificate and attribute certificates.

```
CertificateTrustTrees ::=   SEQUENCE OF CertificateTrustPoint

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

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

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

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

The acceptablePolicySet field identifies the initial set of certificate policies, any of which are acceptable under the signature policy. AcceptablePolicySet ::= SEQUENCE OF CertPolicyId

```
CertPolicyId ::= OBJECT IDENTIFIER
```

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

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

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

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

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

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

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

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

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

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

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

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

3.6.2 Revocation Requirements

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

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

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

* endCertRevReq: end certificates (i.e., the signers certificate, the attribute certificate or the time-stamping authority certificate).

* caCerts: CA certificates.

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

An authority certificate is certificate issued to an authority (e.g., either to a certification authority or to an attribute authority (AA)).

A Time-Stamping Authority (TSA) is a trusted third party that creates time-stamp tokens in order to indicate that a datum existed at a particular point in time. See [TSP].

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

Revocation requirements are specified in terms of:

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

3.7 Signing Certificate Trust Conditions

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

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

3.8 Time-Stamp Trust Conditions

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

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

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

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

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

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

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

The signatureTimestampDelay field specifies a maximum acceptable time between the signing time and the time at which the signature time-stamp, as used to form the ES Time-Stamped, is created for the

verifier. If the signature time-stamp is later that the time in the signing-time attribute by more than the value given in signatureTimestampDelay, the signature must be considered invalid.

3.9 Attribute Trust Conditions

If the attributeTrustCondition field is not present then any certified attributes may not considered to be valid under this validation policy. The AttributeTrustCondition field is defined as follows:

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

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

The howCertAttribute field specifies whether attributes uncertified attributes "claimed" by the signer, or certified attributes (i.e., Attribute Certificates) or either using the signer attributes attribute defined in [ES-FORMATS] section 3.12.3.

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

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

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

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

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

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

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

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

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

3.10 Algorithm Constraints

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

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

AlgorithmConstraints ::= SEQUENCE OF AlgAndLength

AlgAndLength ::= SEQUENCE {
        algID                   OBJECT IDENTIFIER,
        minKeyLength    INTEGER         OPTIONAL,
       -- Minimum key length in bits
        other           SignPolExtensions OPTIONAL
                }
```

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

3.11 Signature Policy Extensions

Additional signature policy rules may be added to:

* the overall signature policy structure, as defined in section 3.1;
* the signature validation policy structure, as defined in section 3.2;
* the common rules, as defined in section 3.3;
* the commitment rules, as defined in section 3.4;
* the signer rules, as defined in section 3.5.1;
* the verifier rules, as defined in section 3.5.2;
* the revocation requirements in section 3.6.2;
* the algorithm constraints in section 3.10.

These extensions to the signature policy rules must be defined using an ASN.1 syntax with an associated object identifier carried in the SignPolExtn as defined below:

```
SignPolExtensions ::= SEQUENCE OF SignPolExtn

SignPolExtn ::= SEQUENCE {
                extnID      OBJECT IDENTIFIER,
                extnValue   OCTET STRING  }
```

The extnID field must contain the object identifier for the extension. The extnValue field must contain the DER (see ITU-T Recommendation X.690 [4]) encoded value of the extension. The definition of an extension, as identified by extnID must include a definition of the syntax and semantics of the extension.

4. Security Considerations

4.1 Protection of Private Key

The security of the electronic signature mechanism defined in this document depends on the privacy of the signer’s private key. Implementations must take steps to ensure that private keys cannot be compromised.

4.2 Choice of Algorithms

Implementers should be aware that cryptographic algorithms become weaker with time. As new cryptoanalysis techniques are developed and computing performance improves, the work factor to break a particular

cryptographic algorithm will reduce. Therefore, cryptographic algorithm implementations should be modular allowing new algorithms to be readily inserted. That is, implementers should be prepared for the set of mandatory to implement algorithms to change over time.

5. Conformance Requirements

Signer and verifier systems shall be able to process an electronic signature in accordance with the specification of the signature policy signature policy referenced identifiable by an Object Identifier, see section 3.

6. References

[TS101733] ETSI Standard TS 101 733 V.1.2.2 (2000-12) Electronic Signature Formats. Note: copies of ETSI TS 101 733 can be freely download from the ETSI web site www.etsi.org.

[ES-FORMATS] Pinkas, D., Ross, J. and N. Pope, "Electronic Signature Formats for Long Term Electronic Signatures", RFC 3126, June 2001.

[TSP] Adams, C, Pinkas, D., Zuccherato, R. and P. Cain, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, August 2001.

[OCSP] Myers, M., Ankney, R., Malpani, R., Galperin, S. and C. Adams, "On-line Status Certificate Protocol", RFC 2560, June 1999.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[ESS] Hoffman, P., "Enhanced Security Services for S/MIME", RFC 2634, June 1999.

[CMS] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[RFC2459] Housley, R., Ford, W., Polk, W. and D. Solo, "Internet X.509 Public Key Infrastructure, Certificate and CRL Profile," RFC 2459, January 1999.

[PKCS9] RSA Laboratories, "The Public-Key Cryptography Standards (PKCS)", RSA Data Security Inc., Redwood City, California, November 1993 Release.

[ISONR] ISO/IEC 10181-5: Security Frameworks in Open Systems. Non-Repudiation Framework. April 1997.

7. Authors' Addresses

This Experimental RFC has been produced in ETSI TC-SEC.

ETSI
F-06921 Sophia Antipolis, Cedex - FRANCE
650 Route des Lucioles - Sophia Antipolis
Valbonne - FranceTel: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16
secretariat@etsi.fr
http://www.etsi.org

Contact Point

Harri Rasilainen
ETSI
650 Route des Lucioles
F-06921 Sophia Antipolis Cedex
FRANCE

EMail: harri.rasilainen@etsi.fr

John Ross
Security & Standards
192 Moulsham Street
Chelmsford, Essex
CM2 0LG
United Kingdom

EMail: ross@secstan.com

Denis Pinkas
Integris, Bull.
68, Route de Versailles
78434 Louveciennes CEDEX
FRANCE

EMail: Denis.Pinkas@bull.net

Nick Pope
Security & Standards
192 Moulsham Street
Chelmsford, Essex
CM2 0LG
United Kingdom
EMail: pope@secstan.com

Annex A (normative):

ASN.1 Definitions This annex provides the reference definition of the ASN.1 syntax signature policies definitions for new syntax defined in this document.

A.1 Definitions Using X.208 (1988) ASN.1 Syntax

NOTE: The ASN.1 Module defined in section A.1 has precedence over that defined in Annex A-2 in the case of any conflict.

```
ETS-ElectronicSignaturePolicies-88syntax { iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-mod(0)
    7}

DEFINITIONS EXPLICIT TAGS ::=
BEGIN
-- EXPORTS All

IMPORTS

-- Internet X.509 Public Key Infrastructure
 - Certificate and CRL Profile: RFC 2560
       Certificate, AlgorithmIdentifier, CertificateList, Name,
       GeneralNames, GeneralName, DirectoryString,Attribute,
       AttributeTypeAndValue, AttributeType, AttributeValue,
        PolicyInformation, BMPString, UTF8String

  FROM PKIX1Explicit88
       {iso(1) identified-organization(3) dod(6) internet(1)
       security(5) mechanisms(5) pkix(7) id-mod(0)
       id-pkix1-explicit-88(1)}
                                              ;

-- Signature Policy Specification
-- ==============================

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

SignPolicyHash ::= OCTET STRING

SignPolicyInfo ::= SEQUENCE {
       signPolicyIdentifier            SignPolicyId,
       dateOfIssue                     GeneralizedTime,
       policyIssuerName                PolicyIssuerName,
```

```
        fieldOfApplication              FieldOfApplication,
        signatureValidationPolicy       SignatureValidationPolicy,
        signPolExtensions               SignPolExtensions
                                              OPTIONAL
                                                      }

PolicyIssuerName ::= GeneralNames

FieldOfApplication ::= DirectoryString

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

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


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

CommitmentRules ::= SEQUENCE OF CommitmentRule

CommitmentRule  ::= SEQUENCE {
        selCommitmentTypes                  SelectedCommitmentTypes,
        signerAndVeriferRules          [0]  SignerAndVerifierRules
                                                OPTIONAL,
        signingCertTrustCondition      [1]  SigningCertTrustCondition
                                                OPTIONAL,
        timeStampTrustCondition        [2]  TimestampTrustCondition
                                                OPTIONAL,
```

```
        attributeTrustCondition        [3]  AttributeTrustCondition
                                                  OPTIONAL,
        algorithmConstraintSet         [4]  AlgorithmConstraintSet
                                                  OPTIONAL,
        signPolExtensions              [5]  SignPolExtensions
                                                   OPTIONAL
                                                         }
SelectedCommitmentTypes ::= SEQUENCE OF CHOICE {
        empty                          NULL,
        recognizedCommitmentType       CommitmentType }

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

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


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

CMSAttrs ::= SEQUENCE OF OBJECT IDENTIFIER

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

                                                      }

CertInfoReq ::= ENUMERATED {
```

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

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

MandatedUnsignedAttr ::=  CMSAttrs
-- Mandated CMS unsigned attributed

CertificateTrustTrees ::=   SEQUENCE OF CertificateTrustPoint

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

PathLenConstraint    ::=   INTEGER (0..MAX)

AcceptablePolicySet ::= SEQUENCE OF CertPolicyId

CertPolicyId ::= OBJECT IDENTIFIER

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

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

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

      BaseDistance ::= INTEGER (0..MAX)

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

```
        inhibitPolicyMapping            [1] SkipCerts OPTIONAL }

SkipCerts ::= INTEGER (0..MAX)

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

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

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

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

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

DeltaTime ::= SEQUENCE {
        deltaSeconds    INTEGER,
        deltaMinutes    INTEGER,
```

```
        deltaHours      INTEGER,
        deltaDays       INTEGER }

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

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

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


AttributeTypeConstraints ::= SEQUENCE OF AttributeType

AttributeValueConstraints ::= SEQUENCE OF AttributeTypeAndValue


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

AlgorithmConstraints ::= SEQUENCE OF AlgAndLength

AlgAndLength ::= SEQUENCE {
        algID                   OBJECT IDENTIFIER,
        minKeyLength    INTEGER         OPTIONAL,
                              -- Minimum key length in bits other
                SignPolExtensions OPTIONAL
```

```
                }

SignPolExtensions ::= SEQUENCE OF SignPolExtn

SignPolExtn ::= SEQUENCE {
        extnID      OBJECT IDENTIFIER,
                extnValue   OCTET STRING  }


END -- ETS-ElectronicSignaturePolicies-88syntax --
```

A.2 Definitions Using X.680 1997 ASN.1 Syntax

NOTE: The ASN.1 module defined in section A.1 has precedence over that defined in section A.2 in the case of any conflict.

```
ETS-ElectronicSignaturePolicies-97Syntax { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-mod(0) 8}

DEFINITIONS EXPLICIT TAGS ::=
BEGIN
-- EXPORTS All -

IMPORTS

-- Internet X.509 Public Key Infrastructure
-- Certificate and CRL Profile: RFC 2560
        Certificate, AlgorithmIdentifier, CertificateList, Name,
        GeneralNames, GeneralName, DirectoryString, Attribute,
        AttributeTypeAndValue, AttributeType, AttributeValue,
        PolicyInformation

  FROM PKIX1Explicit93
        {iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        nid-pkix1-explicit-88(1)}
;

-- S/MIME Object Identifier arcs used in the present document
-- ==========================================================

-- S/MIME  OID arc used in the present document
-- id-smime OBJECT IDENTIFIER ::= { iso(1) member-body(2)
--             us(840) rsadsi(113549) pkcs(1) pkcs-9(9) 16 }

-- S/MIME Arcs
-- id-mod  OBJECT IDENTIFIER ::= { id-smime 0 }
-- modules
```

```
-- id-ct   OBJECT IDENTIFIER ::= { id-smime 1 }
-- content types
-- id-aa   OBJECT IDENTIFIER ::= { id-smime 2 }
-- attributes
-- id-spq  OBJECT IDENTIFIER ::= { id-smime 5 }
-- signature policy qualifier
-- id-cti  OBJECT IDENTIFIER ::= { id-smime 6 }
-- commitment type identifier
-- Signature Policy Specification
-- ==============================

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

SignPolicyHash ::= OCTET STRING

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

SignPolicyId ::= OBJECT IDENTIFIER

PolicyIssuerName ::= GeneralNames

FieldOfApplication ::= DirectoryString

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

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

CommonRules  ::= SEQUENCE {
        signerAndVeriferRules          [0]  SignerAndVerifierRules
                                                OPTIONAL,
```

```
        signingCertTrustCondition      [1]  SigningCertTrustCondition
                                              OPTIONAL,
        timeStampTrustCondition        [2]  TimestampTrustCondition
                                              OPTIONAL,
        attributeTrustCondition        [3]  AttributeTrustCondition
                                              OPTIONAL,
        algorithmConstraintSet         [4]  AlgorithmConstraintSet
                                              OPTIONAL,
        signPolExtensions              [5]  SignPolExtensions
                                              OPTIONAL
                                            }

CommitmentRules ::= SEQUENCE OF CommitmentRule

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

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

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

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

SignerRules ::= SEQUENCE {
        externalSignedData         BOOLEAN      OPTIONAL,
                     -- True if signed data is external to CMS structure
                        -- False if signed data part of CMS structure
                        -- not present if either allowed
```

```
        mandatedSignedAttr         CMSAttrs,
                  -- Mandated CMS signed attributes
        mandatedUnsignedAttr       CMSAttrs,
                  -- Mandated CMS unsigned attributed
        mandatedCertificateRef     [0] CertRefReq DEFAULT signerOnly,
                        -- Mandated Certificate Reference
        mandatedCertificateInfo    [1] CertInfoReq DEFAULT none,
                        -- Mandated Certificate Info
        signPolExtensions               [2] SignPolExtensions  OPTIONAL
                                                    }

CMSAttrs ::= SEQUENCE OF OBJECT IDENTIFIER

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

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

VerifierRules ::= SEQUENCE {
                        mandatedUnsignedAttr    MandatedUnsignedAttr,
                        signPolExtensions       SignPolExtensions  OPTIONAL
                        }
MandatedUnsignedAttr ::=  CMSAttrs
                                -- Mandated CMS unsigned attributed

CertificateTrustTrees ::=   SEQUENCE OF CertificateTrustPoint

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

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

AcceptablePolicySet ::= SEQUENCE OF CertPolicyId

CertPolicyId ::= OBJECT IDENTIFIER

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

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

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

      BaseDistance ::= INTEGER (0..MAX)

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

SkipCerts ::= INTEGER (0..MAX)

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

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

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

```
        other           (5)
                    -- Other mechanism as defined by signature policy
                    -- extension
                                            }

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


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

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

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

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

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

```
AttributeTypeConstraints ::= SEQUENCE OF AttributeType

AttributeValueConstraints ::= SEQUENCE OF AttributeTypeAndValue

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

AlgorithmConstraints ::= SEQUENCE OF AlgAndLength

AlgAndLength ::= SEQUENCE {
        algID           OBJECT IDENTIFIER,
        minKeyLength    INTEGER         OPTIONAL,
                                -- Minimum key length in bits
        other           SignPolExtensions OPTIONAL
                                                        }

SignPolExtensions ::= SEQUENCE OF SignPolExtn

SignPolExtn ::= SEQUENCE {
        extnID      OBJECT IDENTIFIER,
        extnValue   OCTET STRING  }

END                             -- ETS-ElectronicPolicies-97Syntax
```

Annex B (informative):

B.1 Signature Policy and Signature Validation Policy

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

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

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

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

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

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

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

This document places no restriction on how it should be implemented. Provide the implementation conforms to the conformance requirements as define in section 5 implementation options include:

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

A validation process that is dynamically programmable and able to adapt its validation rules in accordance with a description of the signature policy provided in a computer-processable language. This present document defines such a policy using an ASN.1 structure (see 6.1). This type of implementation could support multiple signature policies without being modified every time, provided all the validation rules specified as part of the signature policy are known by the implementation. (i.e., only requires modification if there are additional rules specified).

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

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

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

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

B.2 Identification of Signature Policy

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

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

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

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

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

B.3 General Signature Policy Information

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

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

* Field of application: This defines in general terms the general legal/contract/application contexts in which the signature policy is to be used and the specific purposes for which the electronic signature is to be applied.

B.4 Recognized Commitment Types

The signature validation policy may recognize one or more types of commitment as being supported by electronic signatures produced under the security policy. If an electronic signature does not contain a recognized commitment type then the semantics of the electronic signature is dependent on the data being signed and the context in which it is being used.

Only recognized commitment types are allowed in an electronic signature.

The definition of a commitment type includes:

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

The definition of a commitment type can be registered:

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

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

B.5 Rules for Use of Certification Authorities

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

B.5.1 Trust Points

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

B.5.2 Certification Path

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

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

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

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

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

Certificate path processing, thus generally starts with one of the trust point from the signature policy and ends with the user certificate. The certificate path processing procedures defined in RFC 2459 section 6 identifies the following initial parameters that are selected by the verifier in certificate path processing:

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

The signature validation policy identifies constraints on these parameters.

B.6 Revocation Rules

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

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

B.7 Rules for the Use of Roles

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

B.7.1 Attribute Values

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

B.7.2 Trust Points for Certified Attributes

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

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

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

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

B.7.3 Certification Path for Certified Attributes

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

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

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

B.8 Rules for the Use of Time-Stamping and Timing

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

B.8.1 Trust Points and Certificate Paths

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

B.8.2 Time-Stamping Authority Names

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

B.8.3 Timing Constraints - Caution Period

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

The validation policy may specify such a cautionary period.

B.8.4 Timing Constraints - Time-Stamp Delay

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

B.9 Rules for Verification Data to be followed

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

These verification data rules should include:

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

B.10 Rules for Algorithm Constraints and Key Lengths

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

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

B.11 Other Signature Policy Rules

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

B.12 Signature Policy Protection

When signer or verifier obtains a copy of the Signature Policy from an issuer, the source should be authenticated (for example by using electronic signatures). When the signer references a signature policy the Object Identifier (OID) of the policy, the hash value and the hash algorithm OID of that policy must be included in the Electronic Signature.

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

The hash value may be obtained by:

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

the signer, having verified the source of the policy, may use both the hash algorithm and the hash value included in the computer processable form of the policy (see section 6.1).

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.