INTERNATIONAL STANDARD

ISO/IEC 9594-8

Eighth edition
2017-05

Information technology — Open Systems Interconnection — The Directory —

Part 8: **Public-key and attribute certificate frameworks**

Technologies de l'information — Interconnexion de systèmes ouverts (OSI) — L'annuaire —

Partie 8: Cadre général des certificats de clé publique et d'attribut

Reference number
ISO/IEC 9594-8:2017(E)

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This eighth edition cancels and replaces the seventh edition (ISO/IEC 9594-8:2014), which has been technically revised.

This document was prepared by ISO/IEC JTC 1, *Information technology*, SC 6, *Telecommunications and information exchange between systems*, in collaboration with ITU-T. The identical text is published as ITU-T X.509 (10/2016).

A list of all parts in the ISO/IEC 9594 series, published under the general title *Information technology — Open Systems Interconnection — The Directory*, can be found on the ISO website.

CONTENTS

Page

Introduction

Many applications have requirements for security to protect against threats to the communication of information. Virtually all security services are dependent upon the identities of the communicating parties being reliably known, i.e., authenticated.

This Recommendation | International Standard defines a framework for public-key certificates. This framework includes the specification of data objects used to represent the public-key certificates themselves, as well as revocation notices for issued public-key certificates that should no longer be trusted. It defines some critical components of a public-key infrastructure (PKI), but it does not define a PKI in its entirety. However, this Recommendation | International Standard provides the foundation upon which full PKIs and their specifications can be built.

Similarly, this Recommendation | International Standard defines a framework for attribute certificates. This framework includes the specification of data objects used to represent the attribute certificates themselves, as well as revocation notices for issued attribute certificates that should no longer be trusted. It defines some critical components of a privilege management infrastructure (PMI), but it does not define a PMI in its entirety. However, this Recommendation | International Standard provides the foundation upon which full PMIs and their specifications can be built.

Schema definitions are defined for holding PKI and PMI information in a directory according to the specification found in the ITU-T X.500 series of Recommendations | ISO/IEC 9594 (all parts) or according to the lightweight directory access protocol (LDAP) specification.

This Recommendation | International Standard provides the foundation frameworks upon which industry profiles can be defined by other standards groups and industry forums. Many of the features defined as optional in these frameworks may be mandated for use in certain environments through profiles. This eighth edition technically revises and enhances the seventh edition of this Recommendation | International Standard.

This eighth edition specifies versions 1, 2 and 3 of public-key certificates, versions 1 and 2 of certificate revocation lists and version 2 of attribute certificates.

The extensibility function was added in an earlier edition with version 3 of the public-key certificate and with version 2 of the certificate revocation list and was incorporated into the attribute certificate from its initial inception.

Annex A, which is an integral part of this Recommendation | International Standard, provides the ASN.1 modules which contain all of the definitions associated with the frameworks.

Annex B, which is not an integral part of this Recommendation | International Standard, lists object identifiers assigned to cryptographic algorithms defined by other specifications. It is provided for easy reference and import into other ASN.1 modules.

Annex C, which is an integral part of this Recommendation | International Standard, provides definitions for how certificate extension types may be represented by directory attribute types.

Annex D, which is not an integral part of this Recommendation | International Standard, includes extracts of external ASN.1 modules referenced by this Recommendation | International Standard.

Annex E, which is an integral part of this Recommendation | International Standard, provides rules for generating and processing certificate revocation lists (CRLs).

Annex F, which is not an integral part of this Recommendation | International Standard, provides examples of delta certificate revocation list (CRL) issuance.

Annex G, which is not an integral part of this Recommendation | International Standard, provides examples of privilege policy syntaxes and privilege attributes.

Annex H, which is not an integral part of this Recommendation | International Standard, is an introduction to public-key cryptography.

Annex I, which is not an integral part of this Recommendation | International Standard, contains examples of the use of certification path constraints.

Annex J, which is not an integral part of this Recommendation | International Standard, provides guidance for public-key infrastructure (PKI) enabled applications on the processing of certificate policy while in the certification path validation process.

Annex K, which is not an integral part of this Recommendation | International Standard, provides guidance on the use of the **`contentCommitment`** bit in the **`keyUsage`** certificate extension.

Annex L, which is not an integral part of this Recommendation | International Standard, includes public-key and attribute certificate extensions that have been deprecated.

Annex M, which is not an integral part of this Recommendation | International Standard, gives a short introduction to directory and distinguished name concepts.

Annex N, which is not an integral part of this Recommendation | International Standard, provides some general considerations on strong authentication.

Annex O, which is not an integral part of this Recommendation | International Standard, contains an alphabetical list of information item definitions in this Recommendation | International Standard.

Annex P, which is not an integral part of this Recommendation | International Standard, lists the amendments and defect reports that have been incorporated to form this edition of this Recommendation | International Standard.

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks

1 Scope

This Recommendation | International Standard addresses some of the security requirements in the areas of authentication and other security services through the provision of a set of frameworks upon which full services can be based. Specifically, this Recommendation | International Standard defines frameworks for:

- public-key certificates; and
- attribute certificates.

The public-key certificate framework defined in this Recommendation | International Standard specifies the information objects and data types for a public-key infrastructure (PKI), including public-key certificates, certificate revocation lists (CRLs), trust broker and authorization and validation lists (AVLs). The attribute certificate framework specifies the information objects and data types for a privilege management infrastructure (PMI), including attribute certificates, and attribute certificate revocation lists (ACRLs). This Recommendation | International Standard also provides the framework for issuing, managing, using and revoking certificates. An extensibility mechanism is included in the defined formats for both certificate types and for all revocation list schemes. This Recommendation | International Standard also includes a set of extensions, which is expected to be generally useful across a number of applications of PKI and PMI. The schema components (including object classes, attribute types and matching rules) for storing PKI and PMI information in a directory, are included in this Recommendation | International Standard.

This Recommendation | International Standard specifies the framework for strong authentication, involving credentials formed using cryptographic techniques. It is not intended to establish this as a general framework for authentication, but it can be of general use for applications which consider these techniques adequate.

Authentication (and other security services) can only be provided within the context of a defined security policy. It is a matter for users of an application to define their own security policy.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- Recommendation ITU-T X.411 (1999) | ISO/IEC 10021-4:2003, *Information technology – Message Handling Systems (MHS) – Message Transfer System: Abstract Service Definition and Procedures*.
- Recommendation ITU-T X.500 (2016) | ISO/IEC 9594-1:2017, *Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services*.
- Recommendation ITU-T X.501 (2016) | ISO/IEC 9594-2:2017, *Information technology – Open Systems Interconnection – The Directory: Models*.
- Recommendation ITU-T X.511 (2016) | ISO/IEC 9594-3:2017, *Information technology – Open Systems Interconnection – The Directory: Abstract service definition*.
- Recommendation ITU-T X.519 (2016) | ISO/IEC 9594-5:2017, *Information technology – Open Systems Interconnection – The Directory: Protocol specifications*.
- Recommendation ITU-T X.520 (2016) | ISO/IEC 9594-6:2017, *Information technology – Open Systems Interconnection – The Directory: Selected attribute types*.
- Recommendation ITU-T X.660 (2011) | ISO/IEC 9834-1:2012, *Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: General procedures and top arcs of the International Object Identifier tree*.

- Recommendation ITU-T X.681 (2015) | ISO/IEC 8824-2:2015, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.
- Recommendation ITU-T X.690 (2015) | ISO/IEC 8825-1:2015, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*.
- Recommendation ITU-T X.812 (1995) | ISO/IEC 10181-3:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Access control framework*.
- Recommendation ITU-T X.813 (1996) | ISO/IEC 10181-4:1997, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Non-repudiation framework*.
- Recommendation ITU-T X.841 (2000) | ISO/IEC 15816:2002, *Information technology – Security techniques – Security information objects for access control*.

2.2 Paired Recommendations | International Standards equivalent in technical content

- Recommendation ITU-T X.800 (1991), *Security architecture for Open Systems Interconnection for CCITT applications*.

  ISO 7498-2:1989, *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture*.

2.3 Recommendations

- Recommendation ITU-T X.1252 (2010), *Baseline identity management terms and definitions*.

2.4 Other references

- IETF RFC 791 (1981), *Internet Protocol*.
- IETF RFC 822 (1982), *Standard for the Format of ARPA Internet Text Messages*.
- IETF RFC 1630 (1994), *Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web*.
- IETF RFC 3492 (2003), *Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)*.
- IETF RFC 4511 (2006), *Lightweight Directory Access Protocol (LDAP): The Protocol*.
- IETF RFC 4523 (2006), *Lightweight Directory Access Protocol (LDAP) Schema Definitions for X.509 Certificates*.
- IETF RFC 5280 (2008), *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.
- IETF RFC 5890 (2010), *Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework*.
- IETF RFC 5914 (2010), *Trust Anchor Format*.
- IETF RFC 6960 (2013), *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*

3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

3.1 OSI Reference Model security architecture definitions

The following terms are defined in Rec. ITU-T X.800 | ISO 7498-2:

a) authentication exchange;

b) authentication information;

c) confidentiality;

d) credentials;

e) cryptography;

f) data origin authentication;

g) decipherment;

h) digital signature;

i) encipherment; and

j) key;

3.2 Baseline identity management terms and definitions

The following term is defined in Rec. ITU-T X.1252:

a) trust.

3.3 Directory model definitions

The following terms are defined in Rec. ITU-T X.501 | ISO/IEC 9594-2:

a) attribute;

c) directory information tree (DIT);

b) directory system agent;

c) directory user agent (DUA);

d) distinguished name;

e) entry;

f) relative distinguished name;

g) root.

3.4 Access control framework definitions

The following terms are defined in Rec. ITU-T X.812 | ISO/IEC 10181-3:

a) access control decision function (ADF);

b) access control enforcement function (AEF).

3.5 Public-key and attribute certificate definitions

The following terms are defined in this Recommendation | International Standard:

3.5.1 ACRL distribution point: A directory entry or another distribution source for **attribute certificate revocation list**s (ACRLs); an ACRL distributed through an ACRL distribution point may contain revocation entries for only a subset of the full set of attribute certificates issued by one attribute authority (AA) or may contain revocation entries for multiple AAs.

3.5.2 attribute authority certificate: An attribute certificate for one attribute authority (AA) issued by another AA or by the same AA.

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

3.5.4 attribute authority (AA): An authority which assigns privileges by issuing attribute certificates or by including them in public-key certificates.

3.5.5 attribute authority revocation list (AARL): A revocation list containing a list of references to attribute certificates issued to attribute authorities (AAs) that are no longer considered valid by the issuing AA.

3.5.6 attribute certificate revocation list (ACRL): A revocation list containing a list of references to attribute certificates that are no longer considered valid by the issuing attribute authority.

3.5.7 attribute certificate validation: The process of ensuring that an attribute certificate was valid at a given time, including possibly the construction and processing of a delegation path, and ensuring that all attribute certificates in that path were valid (e.g., were not expired or revoked) at that given time.

3.5.8 authority: An entity, responsible for the issuance of certificates or of revocation lists. Four types are defined in this Recommendation | International Standard; a certification authority which issues public-key certificates, an attribute authority which issues attribute certificates, a certificate revocation list (CRL) issuer which issues CRLs and an attribute certificate revocation list (ACRL) issuer which issues ACRLs.

3.5.9 authorization and validation list (AVL): A signed list containing information to an AVL entity about potential communications entities and possible restrictions on the communications with such entities.

3.5.10 authorization and validation list entity (AVL entity): An entity, when acting as a relying party, is dependent on an AVL issued by a designated authorizer.

3.5.11 authorizer: An entity trusted by one or more entities operating as authorization and validation list (AVL) entities to create, maintain and sign authorization and validation lists.

3.5.12 base attribute certificate revocation list (base ACRL): An attribute certificate revocation list (ACRL) that is used as the foundation in the generation of a delta attribute certificate revocation list (dACRL).

3.5.13 base certificate revocation list (base CRL): A certificate revocation list (CRL) that is used as the foundation in the generation of a delta certificate revocation list (dCRL).

3.5.14 CA certificate: A public-key certificate for one certification authority (CA) issued by another CA or by the same CA.

3.5.15 certificate policy: A named set of rules that indicates the applicability of a public-key certificate to a particular community and/or class of application with common security requirements. For example, a particular certificate policy might indicate applicability of a type of certificate to the authentication of electronic data interchange transactions for the trading of goods within a given price range.

3.5.16 certification practice statement (CPS): A statement of the practices that a certification authority (CA) employs in issuing certificates.

3.5.17 certificate revocation list (CRL): A signed list indicating a set of public-key certificates that are no longer considered valid by the issuing certification authority (CA). In addition to the generic term certificate revocation list (CRL), some specific CRL types are defined for CRLs that cover particular scopes.

3.5.18 certificate serial number: An integer value, unique within the issuing authority, which is unambiguously associated with a certificate issued by that authority.

3.5.19 certification authority (CA): An authority trusted by one or more entities to create and digital sign public-key certificates. Optionally the certification authority may create the subjects' keys.

3.5.20 certification authority revocation list (CARL): A revocation list containing a list of certification authority (CA) certificates issued to CAs that are no longer considered valid by the issuing CA.

3.5.21 certification path: An ordered list of one or more public-key certificates, starting with a public-key certificate signed by the trust anchor, and ending with the end-entity public-key certificate to be validated. All intermediate public-key certificates, if any, are certification authority (CA) certificates in which the subject of the preceding public-key certificate is the issuer of the following public-key certificate.

3.5.22 CRL distribution point: A directory entry or other distribution source for certificate revocation lists (CRLs); a CRL distributed through a CRL distribution point may contain revocation entries for only a subset of the full set of public-key certificates issued by one certification authority (CA) or may contain revocation entries for multiple CAs.

3.3.23 cross-certificate: A certification authority (CA) certificate where the issuer and the subject are different CAs. CAs issue cross-certificates to other CAs as a mechanism to authorize the subject CA's existence.

3.5.24 delegation: Conveyance of privilege from one entity that holds such privilege, to another entity.

3.5.25 delegation path: An ordered sequence of attribute certificates which, together with authentication of a privilege asserter's identity can be processed to verify the authenticity of a privilege asserter's privilege.

3.5.26 delta attribute certificate revocation list (dACRL): A partial attribute certificate revocation list that only contains entries for attribute certificates that have had their revocation status changed since the issuance of the referenced base attribute certificate revocation list (base ACRL).

3.4.27 delta-CRL (dCRL): A partial revocation list that only contains entries for public-key certificates that have had their revocation status changed since the issuance of the referenced base CRL.

3.5.28 end entity: Either a public-key certificate subject that uses its private key for purposes other than signing public-key certificates, or an attribute certificate holder that cannot delegate privileges of the attribute certificate, but uses its attributes only to gain access to a resource.

3.5.29 end-entity attribute certificate: An attribute certificate issued to an entity, which then acts as an end entity within a privilege management infrastructure.

3.5.30 end-entity attribute certificate revocation list (EARL): A revocation list containing a list of attribute certificates issued to holders that are not also attribute authorities (AAs) that are no longer considered valid by the certificate issuer.

3.5.31 end-entity public-key certificate: A public-key certificate issued to an entity, which then acts as an end entity within a public-key infrastructure.

3.5.32 end-entity public-key certificate revocation list (EPRL): A revocation list containing a list of public-key certificates issued to subjects that are not also certification authorities (CAs), that are no longer considered valid by the certificate issuer.

3.5.33 environmental variables: Those aspects of policy required for an authorization decision, that are not contained within static structures, but are available through some local means to a privilege verifier (e.g., time of day or current account balance).

3.5.34 full attribute certificate revocation list (full ACRL): A complete revocation list that contains entries for all attribute certificates that have been revoked for a given scope.

3.5.35 full CRL: A complete revocation list that contains entries for all public-key certificates that have been revoked for the given scope.

3.5.36 hash function: A (mathematical) function which maps data of arbitrary size into data of a fixed size called a digest.

3.5.37 holder: An entity to whom some privilege has been delegated either directly from the source of authority or indirectly through another attribute authority.

3.5.38 indirect attribute certificate revocation list (iACRL): A revocation list whose scope includes attribute certificates issued by one or more attribute authorities (AAs) other than the issuer of the revocation list. The same indirect ACRL is also authoritative for the attribute certificates, if any, issued by the ACRL issuer.

3.5.39 indirect certificate revocation list (iCRL): A revocation list whose scope includes public-key certificates issued by one or more certification authorities (CAs) other than the issuer of the revocation list. The same indirect certificate revocation list (CRL) is also authoritative for the public-key certificates, if any, issued by the CRL issuer.

3.5.40 intermediate CA: A certification authority (CA) is acting as an intermediate CA within a certification path when it is the issuer of the next public-key certificate on that certification path.

3.5.41 key agreement: A method for negotiating a symmetric key value on-line without transferring the key, even in an encrypted form, e.g., the Diffie-Hellman technique (see ISO/IEC 11770-1 for more information on key agreement mechanisms).

3.5.42 object method: An action that can be invoked on a resource (e.g., a file system may have read, write and execute object methods).

3.5.43 one-way function: A (mathematical) function which is easy to compute, but which for a general value *y* in the range, it is computationally difficult to find a value *x* in the domain such that f(*x*) = *y*. There may be a few values *y* for which finding *x* is not computationally difficult.

3.5.44 PKI end entity: An entity that is acting as an end entity in a public-key infrastructure (PKI) environment, where the subject uses it private key for other purposes than signing public-key certificates.

3.5.45 PMI end entity: An entity that is acting as an end entity in a privilege management infrastructure (PMI) environment, where the holder uses its privilege attributes to gain access to a resource.

3.5.46 policy decision point (PDP): The point where policy decisions are made (synonymous with access control decision function, ADF).

3.5.47 policy enforcement point (PEP): The point where the policy decisions are actually enforced (synonymous with access control enforcement function, AEF).

3.5.48 policy mapping: Recognizing that, when a certification authority (CA) in one domain certifies a CA in another domain, a particular certificate policy in the second domain may be considered by the authority of the first domain to be equivalent (but not necessarily identical in all respects) to a particular certificate policy in the first domain.

3.5.49 private key: (In a public-key cryptosystem) that key of an entity's key pair which is known only by that entity.

3.5.50 privilege: An attribute or property assigned to an entity by an attribute authority.

3.5.51 privilege asserter: A privilege holder using the attributes of the attribute certificate or public-key certificate to assert privilege.

3.5.52 privilege holder: An entity that has been assigned privilege. A privilege holder may assert its privilege for a particular purpose.

3.5.53 privilege management infrastructure (PMI): The infrastructure able to support the management of privileges in support of a comprehensive authorization service and in relationship with a public-key infrastructure.

3.5.54 privilege policy: The policy that outlines conditions for privilege verifiers to provide/perform sensitive services to/for qualified privilege asserters. Privilege policy relates attributes associated with the service as well as attributes associated with privilege asserters.

3.5.55 privilege verifier: An entity verifying attributes of a public-key certificate or attribute certificate against a privilege policy.

3.5.56 public key: That key of an entity's key pair which is publicly known.

3.5.57 public-key certificate: The public key of an entity, together with some other information, rendered unforgeable by digital signature with the private key of the certification authority (CA) that issued it.

3.5.58 public-key certificate validation: The process of ensuring that a public-key certificate was valid at a given time, including possibly the construction and processing of a certification path, and ensuring that all public-key certificates in that path were valid (e.g., were not expired or revoked) at that given time.

3.5.59 public-key infrastructure (PKI): The infrastructure able to support the management of public keys able to support authentication, encryption, integrity or non-repudiation services.

3.5.60 registration authority (RA): Those aspects of the responsibilities of a certification authority that are related to identification and authentication of the subject of a public-key certificate to be issued by that certification authority. An RA may either be a separate entity or be an integrated part of the certification authority.

NOTE – This definition is different in scope from the one defined in Rec. ITU-T X.660 | ISO/IEC 9834-1.

3.5.61 relying party: An entity that relies on the data in a public-key certificate in making decisions.

3.5.62 role assignment certificate: A certificate that contains the role attribute, assigning one or more roles to the certificate subject/holder.

3.5.63 role specification certificate: An attribute certificate that contains the assignment of privileges to a role.

3.5.64 sensitivity: Characteristic of a resource that implies its value or importance.

3.5.65 security policy: The set of rules laid down by the security authority governing the use and provision of security services and facilities.

3.5.66 self-issued attribute certificate: An attribute certificate where the issuer and the holder are the same attribute authority (AA). An AA might use a self-issued attribute certificate, for example, to publish policy information.

3.5.67 self-issued certificate: A certification authority (CA) certificate where the issuer and the subject are the same CA. A CA might use self-issued certificates, for example, during a key rollover operation to provide trust from the old key to the new key.

3.5.68 self-signed certificate: A special case of self-issued certificates where the private key used by the certification authority (CA) to sign the CA certificate corresponds to the public key that is certified within the CA certificate. A CA might use a self-signed certificate, for example, to advertise their public key or other information about their operations.

NOTE – Use of self-issued certificates and self-signed certificates issued by other than certification authorities (CAs) are outside the scope of this Recommendation | International Standard.

3.5.69 source of authority (SOA): An attribute authority that a privilege verifier for a particular resource trusts as the ultimate authority to assign a set of privileges for asserting that resource.

3.5.70 subject CA: A certification authority (CA) for which another CA has issued a CA certificate.

3.5.71 trust anchor: An entity that is trusted by a relying party and used for validating public-key certificates.

3.5.72 trust anchor information: Trust anchor information is at least the: distinguished name of the trust anchor, associated public key, algorithm identifier, public key parameters (if applicable), and any constraints on its use, including a validity period. The trust anchor information may be provided as a self-signed certification authority (CA) certificate or as a normal CA certificate (i.e., cross-certificate).

3.5.73 trust anchor store: A trust anchor information collection at a relying party for one or more trust anchors.

3.5.74 trust broker: A third party trusted by relying parties to provide quantitative information about the trustworthiness and qualitative information about the intended use of a public-key certificate based on information about the certification authority. Trust brokers are independent of certification authorities and have direct trust relationships with relying parties.

3.5.75 trust in a CA: Belief that the certification authority (CA) will act reliability and truthfully in the management of its public-key certificates and will comply with its published certification practise statement and relevant legislation.

3.5.76 trust in a public-key certificate: Belief that the public-key certificate belongs to the subject identified in the public-key certificate.

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply:

AA	Attribute Authority
AARL	Attribute Authority Revocation List
ACRL	Attribute Certificate Revocation List
ADF	Access control Decision Function
AEF	Access control Enforcement Function
AIA	Authority Information Access
AVL	Authorization and Validation List
AVMP	Authorization Validation Management Protocol
BER	Basic Encoding Rules
CA	Certification Authority
CARL	Certification Authority Revocation List
CASP	Certification Authority Subscription Protocol
CRL	Certificate Revocation List
dACRL	Delta Attribute Certificate Revocation List
DAP	Directory Access Protocol
dCRL	Delta Certificate Revocation List
DIT	Directory Information Tree
DER	Distinguished Encoding Rules
DS	Delegation Service
DSA	Digital Signature Algorithm
DUA	Directory User Agent
EARL	End-entity Attribute certificate Revocation List
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EPRL	End-entity Public-key certificate Revocation List
FQDN	Fully Qualified Domain Name
HSM	Hardware Security Module
iACRL	Indirect Attribute Certificate Revocation List
iCRL	Indirect Certificate Revocation List
IDN	Internationalized Domain Name
IDP	Issuing Distribution Point
LDAP	Lightweight Directory Access Protocol
LDH	Letters, Digits, Hyphen
M2M	Machine-to-Machine
OCSP	Online Certificate Status Protocol
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIN	Personal Identification Number
PKCS	Public-Key Cryptography Standards

PKI	Public-Key Infrastructure
PMI	Privilege Management Infrastructure
RA	Registration Authority
RDN	Relative Distinguished Name
RoA	Recognition of Authority
RSA	Rivest-Shamir-Adelman
SOA	Source Of Authority
TB	Trust Broker

5 Conventions

The term "Specification" (as in "this Specification") shall be taken to mean Rec. ITU-T X.509 | ISO/IEC 9594-8.

The term "The Directory Specifications" shall be taken to mean the other parts of the ITU-T X.500 series of Recommendations | ISO/IEC 9594 (all parts), excluding this Specification.

This Specification presents ASN.1 notation in the bold Courier New typeface. When ASN.1 types and values are referenced in normal text, they are differentiated from normal text by presenting them in the bold Courier New typeface. When a definition is referenced for the first time in normal text it is presented in italicized Times New Roman.

If the items in a list are numbered (as opposed to using "–" or letters), then the items shall be considered steps in a procedure.

6 Frameworks overview

This Specification defines a framework for obtaining and trusting a public key of an entity in order to verify the digital signature of that entity. If the key pair is of a type that allows encryption and decryption, the public key may also be used for encrypting information to be decrypted by the entity owning the private key. The framework includes the issuance of a public-key certificate by a certification authority (CA) and the validation of that public-key certificate by the relying party, i.e., the entity relying on the content of the public-key certificate. The validation includes:

– establishing a trusted path of public-key certificates between a trusted entity called a trust anchor (see clause 7.5) and the public-key certificate subject, i.e., the entity for which the public-key certificate has been issued;
– verifying the digital signatures on each public-key certificate in the path; and
– validating all the public-key certificates along that path (i.e., that they obey restrictions, were not expired or not revoked at a given time); and
– optionally, asking a trust broker if the public-key certificate can be trusted for the intended purpose.

The public-key infrastructure (PKI) is the infrastructure established to support the issuing, revocation and validation of public-key certificates.

This Specification defines a framework for obtaining and trusting privilege attributes of an entity in order to determine whether they are authorized to access a particular resource. The framework includes the issuance of an attribute certificate by an attribute authority (AA) and the validation of that attribute certificate by a privilege verifier. The validation includes:

– ensuring that the privileges in the attribute certificate are sufficient when compared against the privilege policy;
– establishing a trusted delegation path of attribute certificates if necessary;
– verifying the digital signature on each attribute certificate in the path;
– ensuring that each issuer was authorized to delegate privileges; and
– validating that the attribute certificates have not expired or been revoked by their issuers.

Privileges are provided in directory attributes as defined by Rec. ITU-T X.501 | ISO/IEC 9594-2.

Public-key certificates may also include directory attributes for providing privileges. Such aspects of privileges carried by public-key certificates are also covered by the attribute certificate framework within Section 3.

The privilege management infrastructure (PMI) is the infrastructure established to support the issuing, revocation and validation of attribute-key certificates.

Although PKI and PMI are separate infrastructures and may be established independently from one another, they are related. This Specification recommends that holders and issuers of attribute certificates be identified within attribute certificates by pointers to their appropriate public-key certificates. Authentication of the attribute certificate issuers and holders, to ensure that entities claiming privilege and issuing privilege are who they claim to be, is done using the normal processes of the PKI to authenticate identities. This authentication process is not duplicated within the attribute certificate framework.

An entity may take one more roles in a PKI and/or a PMI. It may act as a CA, as an AA, as an end entity in a PKI environment (PKI end entity), as an end entity in a PMI environment (PMI end entity), as a relying party, etc.

A PKI end entity is an entity that has been assigned an end-entity public-key certificate, where the private key cannot be used to sign other public-key certificates, but may be used for signing for other purposes. A PMI end entity is an entity that uses its end-entity attribute certificate to assess privilege, but it cannot be used for delegating such privilege to other entities.

This Specification makes extensive use of cryptographic algorithms, such as hashing algorithms, public-key algorithms and digital signature algorithms. Annex H introduces these algorithms.

6.1 Digital signatures

This subclause describes digital signatures in general. Sections 2 and 3 of this Specification discuss the use of digital signatures within PKI and PMI specifically. This subclause is not intended to specify a specification for digital signatures in general, but to specify the means by which instances of the PKI and PMI specific data types are signed.

There are different types of digital signatures, such as Rivest-Shamir-Adelman (RSA) digital signatures, digital signature algorithm (DSA) digital signatures and elliptic curve digital signature algorithm (ECDSA) digital signatures.

NOTE 1 – It is not within the scope of this Specification to describe these different techniques in detail, but Annex F gives a short introduction with references to more detailed specifications.

Figure 1 illustrates how a signer of instances of PKI/PMI data types (public-key certificates, attribute certificates, revocation lists, etc.) creates a digital signature and then adds that to the data before transmission. It also illustrates how the recipient of the signed data (the validator) validates the digital signature.

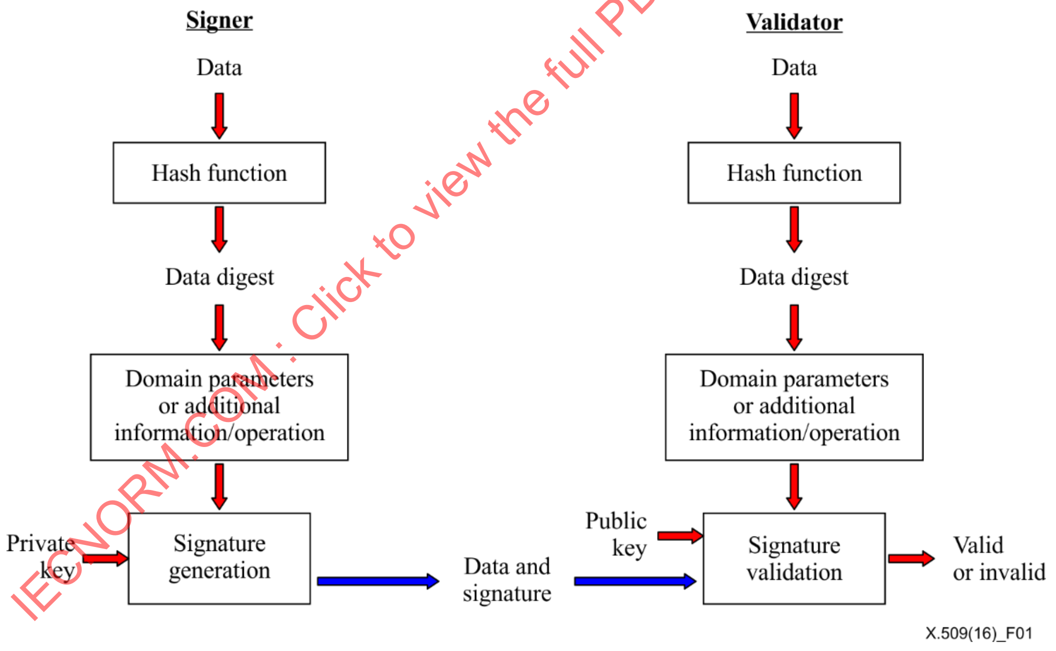

Figure 1 – Digital signature generation and validation

The digital signature signer goes through the following procedure:

1) The signer creates a hash digest over the PKI/PMI data using a secure hashing algorithm (see annex F).
2) The hash digest is then supplemented by additional information in preparation for generation of the digital signature for improved security and for padding the hash digest to a length required by the asymmetric cryptographic function. For the RSA algorithms, that supplementation can be the addition of some information to the hash digest and in some cases, to perform yet another hashing operation. For the DSA and ECDSA signature algorithms, additional domain parameters are added.

3) The result from item 2) together with the private key of the signer and the use of a specific algorithm result in a bit string that together with the used algorithm constitute the digital signature.

4) The signature is appended to data to be signed.

Having received the data, the recipient (validator) goes through a similar procedure:

1) The validator goes through the same procedure as in steps 1) and 2) above, and if the received data is unmodified, the result will be the same as for the signer. If not, the next step will fail.

2) From the result from item 1) together with the public key of the signer, the bit string of the signature and the use of an associated algorithm, the digital signature is evaluated as either valid or invalid.

If the digital signature proves valid, the validator has ensured that the data has not been modified and that the signer is in the position of the private key that corresponds to the public key used by the validator, i.e., the digital signature provides insurance of data integrity and authentication of the signer.

If the digital signature proves invalid, either the data has been modified or the signing private key does not corresponds to the public key used by the validator.

NOTE 2 – Data to be stored in a database or a directory may also be appended a digital signature, which then can evaluated at the retrieval of the data.

6.2 Public-key cryptography and cryptographic algorithms

6.2.1 Formal specification of public-key cryptography

The digital signature of a data item may expressed by the following ASN.1 data type, where the **signature** component is a bit string resulting from using the appropriate signature algorithm.

```
SIGNATURE ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}},
  signature            BIT STRING,
  ... }
```

If a signature is appended to the data, the following ASN.1 may be used to define the data type resulting from applying a signature to the given data type.

```
SIGNED{ToBeSigned} ::= SEQUENCE {
  toBeSigned    ToBeSigned,
  COMPONENTS OF SIGNATURE,
  ... }
```

The following data type is no longer used anymore by this Specification. It may be useful in other areas and is retained for possible import by referencing specifications.

```
HASH{ToBeHashed} ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}},
  hashValue            BIT STRING (CONSTRAINED BY {
   -- shall be the result of applying a hashing procedure to the DER-encoded
   -- octets of a value of -- ToBeHashed } ),
  ... }
```

The following data types are deprecated and are no longer used by this Specification. They are retained for possible import by referencing specifications.

```
ENCRYPTED{ToBeEnciphered} ::= BIT STRING (CONSTRAINED BY {
  -- shall be the result of applying an encipherment procedure
  -- to the BER-encoded octets of a value of -- ToBeEnciphered } )

ENCRYPTED-HASH{ToBeSigned} ::= BIT STRING (CONSTRAINED BY {
  -- shall be the result of applying a hashing procedure to the DER-encoded (see 6.2)
  -- octets of a value of -- ToBeSigned -- and then applying an encipherment procedure
  -- to those octets -- } )
```

6.2.2 Formal definitions of cryptographic algorithms

The following ASN.1 information object class is used for specifying cryptographic algorithms.

```
ALGORITHM ::= CLASS {
  &Type          OPTIONAL,
  &id            OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
  [PARMS        &Type]
  IDENTIFIED BY &id }
```

The following general parameterized data type specifies the syntax of an algorithm specification:

```
AlgorithmIdentifier{ALGORITHM:SupportedAlgorithms} ::= SEQUENCE {
  algorithm    ALGORITHM.&id({SupportedAlgorithms}),
  parameters   ALGORITHM.&Type({SupportedAlgorithms}{@algorithm}) OPTIONAL,
  ... }
```

The **algorithm** component shall be an object identifier that uniquely identifies the cryptographic algorithm being defined.

The **parameters** component, when present, shall specify the parameters associated with the algorithm. Some, but not all algorithms require associated parameters.

```
/* The definitions of the following information object set is deferred to referencing
specifications having a requirement for specific information object sets.*/

SupportedAlgorithms ALGORITHM ::= {...}
```

The following parameterized data type specifies the syntax for fingerprint algorithms:

```
FingerPrint {ToBeFingerprinted} ::= SEQUENCE {
  algorithmIdentifier   AlgorithmIdentifier{{SupportedAlgorithms}},
  fingerprint           BIT STRING,
  ... }
```

Typically, the algorithm used for generating a fingerprint is a hash algorithm, but the above definition is open to other types of algorithms.

The elliptic curve algorithms use the **parameters** component to specify the object identifier identifying a particular curve. The following object gives a general specification for an elliptic curve cryptography (ECC) public key algorithm:

```
ecPublicKey ALGORITHM ::= {
  PARMS        CURVE.&id({SupportedCurves})
  IDENTIFIED   BY id-ecPublicKey }

/* The definitions of the following information value set is deferred to referencing
specifications having a requirement for specific value sets.*/

SupportedCurves OBJECT IDENTIFIER ::= {dummyCurv, ...}

dummyCurv OBJECT IDENTIFIER ::= {2 5 5}
```

Annex B lists some cryptographic algorithms specified in other specification for easy reference.

6.3 Distinguished encoding of basic encoding rules

In order to enable the validation of **SIGNED** and **SIGNATURE** types in a distributed environment, either the original signed message or, in its absence, a canonical encoding is required. The canonical encoding that shall be used by this specification is called the distinguished encoding of a **SIGNED** or **SIGNATURE** data value and shall be obtained by applying the basic encoding rules defined in Rec. ITU-T X.690 | ISO/IEC 8825-1, with the following restrictions:

a) the definite form of length encoding shall be used, encoded in the minimum number of octets;

b) for string types, the constructed form of encoding shall not be used;

c) if the value of a type is its default value, it shall be absent;

d) the components of a Set type shall be encoded in ascending order of their tag values;

e) the components of a Set-of type shall be encoded in ascending order of their octet values;

f) if the value of a Boolean type is **TRUE**, the encoding shall have its contents octet set to "FF";

g) each unused bit in the final octet of the encoding of a Bit String value, if there are any, shall be set to zero;

h) the encoding of a Real type shall be such that bases 8, 10 and 16 shall not be used, and the binary scaling factor shall be zero;

i) the encoding of a UTCTime shall be as specified in Rec. ITU-T X.690 | ISO/IEC 8825-1;

j) the encoding of a GeneralizedTime shall be as specified in Rec. ITU-T X.690 | ISO/IEC 8825-1.

NOTE – These restrictions are the same as the distinguished encoding rules specified in Rec. ITU-T X.690 | ISO/IEC 8825-1.

6.4 Applying distinguished encoding

Generating distinguished encoding requires the abstract syntax of the data to be encoded to be fully understood. An entity may be required to sign data or check the signature of data that is already signed or contains unknown protocol extensions or unknown attribute syntaxes. The entity shall follow these rules:

a) It shall preserve the encoding of received information whose abstract syntax it does not fully know and which it expects to subsequently sign.

b) When signing data for sending, it shall send data whose syntax it fully knows with a distinguished encoding and any other data with its preserved encoding, and shall sign the actual encoding it sends.

c) When checking signatures in received data, it shall check the signature against the actual data received rather than its conversion of the received data to a distinguished encoding.

6.5 Using repositories

PKI and PMI data objects of different kind need to be made available in in some kind of repository. This Specification does not mandate a specific repository technology, but describes repository requirements in term of directories as specified by the Directory Specifications or by the lightweight directory access protocol (LDAP) as specified by IETF RFC 4511.

This Specification also uses directory related terms, like distinguished name and subtree.

A short introduction to directory concepts is given in Annex M.

SECTION 2 – PUBLIC-KEY CERTIFICATE FRAMEWORK

7 Public keys and public-key certificates

7.1 Introduction

The public-key certificate framework defined here is for use by applications with requirements for authentication, integrity, confidentiality and non-repudiation.

The binding of a public-key to an entity is provided by an authority through a digitally signed data structure called a public-key certificate issued by an authority called a certification authority (CA). The format of public-key certificates is defined here, including an extensibility mechanism and a set of specific public-key certificate extensions. If, for any reason, a CA revokes a previously issued public-key certificate, entities need to be able to learn that revocation has occurred so they do not use an untrustworthy public-key certificate. Revocation lists are one scheme that can be used to notify entities of revocations. The format of revocation lists is defined here, including an extensibility mechanism and a set of revocation list extensions. In both the public-key certificate and the revocation list case, other bodies may also define additional extensions that are useful to their specific environments.

An entity that makes decisions based on the validity of a public-key certificates is called a relying party. A relying party needs to validate a public-key certificate prior to using it for a particular transaction in an application. Procedures for performing validation are also defined by this Specification, including verifying the integrity of the public-key certificate itself, its revocation status, and its validity with respect to the intended use.

In order for a relying party to be able to trust a public-key of another entity, for instance to authenticate the identity of that entity, the public key shall be obtained from a trusted source. Such a source, called a certification authority (CA), certifies a public key by issuing a public-key certificate which binds the public-key to the entity which holds the corresponding private key. The procedures used by a CA to ensure that an entity is in fact in possession of the private key and other procedures related to the issuance of public-key certificates are outside the scope of this Specification. However, relying parties should review these procedures and make trust decisions based upon them. The public-key certificate, the form of which is specified later in this clause, has the following properties:

– any relying party with access to the public key of the CA can recover the public key which was certified;

– no party other than the CA can modify the public-key certificate without this being detected (public-key certificates are unforgeable provided that secure cryptographic algorithms are being used in a secure way).

Because public-key certificates are intended to be unforgeable, they can be published by being placed in a directory, without the need for the latter to make special efforts to protect them.

NOTE – Although the CAs are unambiguously defined by distinguished names having a hierarchical structure, this does not imply that there is any relationship between these distinguished names and the hierarchy of the CAs.

7.2 Public-key certificate

A CA issues a public-key certificate of an entity by digital signing (see clause 6.1) a collection of information, including its distinguished name, the entity's distinguished name, a validity period, the value of a public-key algorithm and public key, as well as an optional additional information like for the permitted usage of the user's public key. The following ASN.1 data type specifies the syntax of public-key certificates:

```
Certificate ::= SIGNED{TBSCertificate}

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

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

CertificateSerialNumber ::= INTEGER

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

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

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

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

Extension ::= SEQUENCE {
  extnId     EXTENSION.&id({ExtensionSet}),
  critical   BOOLEAN DEFAULT FALSE,
  extnValue  OCTET STRING
    (CONTAINING EXTENSION.&ExtnType({ExtensionSet}{@extnId})
       ENCODED BY der),
  ... }

der OBJECT IDENTIFIER ::=
  {joint-iso-itu-t asn1(1) ber-derived(2) distinguished-encoding(1)}

ExtensionSet EXTENSION ::= {...}
```

Before a value of **Time** is used in any comparison operation, e.g., as part of a matching rule in a search, and if the syntax of **Time** has been chosen as the **UTCTime** type, the value of the two digit year component shall be rationalized into a four digit year value as follows:

– If the 2-digit value is 00 through to 49 inclusive, the value shall have 2000 added to it.

– If the 2-digit value is 50 through to 99 inclusive, the value shall have 1900 added to it.

NOTE 1 – The use of **GeneralizedTime** may prevent interworking with implementations unaware of the possibility of choosing either **UTCTime** or **GeneralizedTime**. It is the responsibility of those specifying the domains in which public-key certificates defined in this Specification will be used, e.g., profiling groups, as to when the **GeneralizedTime** may be used. In no case shall **UTCTime** be used for representing dates beyond 2049.

The **TBSCertificate** data type is the unsigned public-key certificate and is referred to as a to-be-signed public-key certificate. It shall be encoded using the distinguished encoding rules (DER).

NOTE 2 – Some specifications may specify that a public-key certificate may be transmitted in a non-DER encoding, i.e., in basic encoding rules (BER) encoding without the DER restrictions, but the signature then has to be generated over a DER encoded value of the **TBSCertificate** data type. An otherwise valid signature will then fail the signature validation if the relying party does not decode the public-key certificate and then DER encode it before validating the signature. It is a local policy decision whether in that case to fail the validation or to re-encode the public-key certificate.

The **version** component shall hold the version of the encoded public-key certificate. If the **extensions** component is present in the public-key certificate, **version** shall be **v3**. If the **issuerUniqueIdentifier** or **subjectUniqueIdentifier** component is present **version** shall be **v2** or **v3**.

The **serialNumber** component shall hold an integer assigned by the CA to the public-key certificate. The value of **serialNumber** shall be unique for each public-key certificate issued by a given CA (i.e., the issuer name and serial number together identify a unique public-key certificate).

NOTE 3 – Serial numbers do not have to be in numeric order.

The **signature** component shall contain the identifier for the signature algorithm used by the CA in signing the public-key certificate (e.g., **sha256WithRSAEncryption**, **sha384WithRSAEncryption**, dsa-**with-sha256**, **ecdsa-with-SHA256**, etc.). It shall be the same value as used in the **algorithmIdentifier** component of the **SIGNATURE** data type when signing the public-key certificate.

NOTE 4 – By including this component, the signature algorithm is protected by the signature.

The **issuer** component shall hold the distinguished name of the CA that issued the public-key certificate. It shall hold a non-empty distinguished name.

The **`validity`** component shall hold the time interval during which the CA warrants that it will maintain information about the status of this public-key certificate.

The **`subject`** component shall identify the entity associated with the public-key found in the **`subjectPublicKey`** component of the **`subjectPublicKeyInfo`** component. If the public-key certificate is an end-entity public-key certificate (see clause 7.4), then the distinguished name may be an empty sequence providing that the **`subjectAltName`** extension is present and is flagged as critical (see clause 9.3.2.1). Otherwise, it shall be a non-empty distinguished name.

The **`subjectPublicKeyInfo`** component consists of two subcomponents:

– the **`algorithm`** subcomponent shall hold the algorithm which this public key is an instance of (e.g., **`rsaEncryption`**, **`dhpublicnumber`**, **`id-dsa`**, **`id-ecPublicKey`**, etc.); and

– the **`subjectPublicKey`** subcomponent shall hold the public key being certified.

The **`issuerUniqueIdentifier`** component is used uniquely to identify an issuer in case of name reuse (deprecated).

The **`subjectUniqueIdentifier`** component is used uniquely to identify a subject in case of name reuse (deprecated).

NOTE 5 – The use of **`issuerUniqueIdentifier`** and the **`subjectUniqueIdentifier`** is deprecated. These components were added because at one time there was some fear of the reuse of distinguished names. IETF RFC 5280 forbids the use of these components.

The **`extensions`** component, when present, shall hold one or more extensions as defined in clause 7.3.

An entity may obtain one or more public-key certificates from one or more CAs.

7.3 Public-key certificate extensions

The **`extensions`** component of a public-key certificate allows for the addition of extensions to the structure without modification to the basic ASN.1 data type. An extension consists of an extension identifier, a criticality flag, and an encoding of a data value of an ASN.1 type associated with the identified extension. For those extensions where ordering of individual extensions within the **`SEQUENCE OF`** is significant, the specification of those individual extensions shall include the rules for the significance of the order therein. When a relying party processing a public-key certificate does not recognize an extension and the criticality flag is **`FALSE`**, it may ignore that extension. If the criticality flag is **`TRUE`**, unrecognized extensions shall cause the structure to be considered invalid, i.e., in a public-key certificate, an unrecognized critical extension shall cause validation of a signature using that public-key certificate to fail. When a relying party recognizes and is able to fully process an extension, then the relying party shall process the extension regardless of the value of the criticality flag. When a relying party recognizes and is able to partially process an extension for which the criticality flag is **`TRUE`**, then its behaviour in the presence of unrecognized elements is extension specific and may be documented in each extension. However, the default behaviour, when not specified specifically for an extension, is to treat the entire extension as unrecognized. If unrecognized elements appear within the extension, and the extension is not flagged as critical, those unrecognized elements shall be ignored according to the rules of extensibility documented in clause 12.2.2 in Rec. ITU-T X.519 | ISO/IEC 9594-5.

Any extension that is flagged as non-critical will cause inconsistent behaviour among relying parties that will process the extension and relying parties that do not recognize the extension and will ignore it. The same may be true for extensions that are flagged as critical, between relying parties that can fully process the extension and those that can partially process the extension, depending upon the extension.

A CA issuing a public-key certificate has three options with respect to an extension:

a) it can exclude the extension from the public-key certificate;

b) it can include the extension and flag it as non-critical;

c) it can include the extension and flag it as critical.

A relying party has three possible actions to take with respect to an extension:

a) if the extension is unrecognized and is flagged as non-critical, the relying party shall ignore the extension and accept the public-key certificate (all other things being equal);

b) if the extension is unrecognized and flagged as critical, the relying party shall consider the public-key certificate as invalid;

c) if the extension is recognized, the relying party shall process the extension and accept or consider the public-key certificate invalid, depending on the content of the extension and the conditions under which processing is occurring (e.g., the current values of the path processing variables).

Some extensions shall always be flagged as critical. In these cases, a relying party that understands the extension processes it; the acceptance/rejection of the public-key certificate is dependent (at least in part) on the content of the extension. A relying party that does not understand the extension shall consider the public-key certificate as invalid.

Some extensions shall always be flagged as non-critical. In these cases, a relying party that understands the extension shall process it and acceptance/rejection of the public-key certificate is dependent (at least in part) on the content of the extension. A relying party that does not understand the extension accepts the public-key certificate (unless factors other than this extension cause it to be rejected).

Some extensions may be flagged either as critical or as non-critical. In these cases, a relying party that understands the extension processes it: the acceptance/rejection of the public-key certificate is dependent (at least in part) on the content of the extension, regardless of the criticality flag. A relying party that does not understand the extension accepts the public-key certificate if the extension is flagged as non-critical (unless factors other than this extension cause it to be rejected) and rejects the public-key certificate if the extension is flagged as critical.

When a CA considers including an extension in a public-key certificate it does so with the expectation that its intent will be adhered to wherever possible. If it is necessary that the content of the extension be considered prior to any reliance on the public-key certificate, a CA shall flag the extension as critical. This is done with the realization that any relying party that does not process the extension will reject the public-key certificate (probably limiting the set of applications that can verify the public-key certificate). The CA may mark certain extensions non-critical to achieve backward compatibility with validation applications that cannot process the extensions. Where the need for backward compatibility and interoperability with validation applications incapable of processing the extensions is more vital than the ability of the CA to reinforce the extensions, then these optionally critical extensions would be flagged as non-critical. It is most likely that CAs would set optionally critical extensions as non-critical during a transition period while the verifiers' public-key certificate processing applications are upgraded to ones that can process the extensions.

Specific extensions may be defined in ITU-T Recommendations | International Standards or by any organization which has a need. The object identifier which identifies an extension shall be defined in accordance with Rec. ITU-T X.660 | ISO/IEC 9834-1. Some extensions for public-key certificates are defined in clause 9 of this Specification. Some extensions related to privilege information as defined in clause 17 may also be included in public-key certificates.

The following information object class is used to define specific extensions.

```
EXTENSION ::= CLASS {
  &id           OBJECT IDENTIFIER UNIQUE,
  &ExtnType }
WITH SYNTAX {
  SYNTAX        &ExtnType
  IDENTIFIED BY &id }
```

7.4 Types of public-key certificates

There are two primary types of public-key certificates, end-entity public-key certificates and CA certificates.

An end-entity public-key certificate is a public-key certificate issued by a CA to an entity acting as a PKI end entity that cannot use the corresponding private key to sign other public-key certificates.

A CA certificate is a public-key certificate issued by a CA to an entity that is also acting as a CA and therefore is capable of issuing and signing public-key certificates. A CA certificate shall include the **`basicConstraints`** extension with the **`cA`** components set to **`TRUE`** (see clause 9.4.2.1).

CA certificates can themselves be categorized by the following types:

– Self-issued certificate – This is a CA certificate where the issuer and the subject are the same CA. A CA might use self-issued certificates, for example, during a key rollover operation to provide trust from the old key to the new key.

– Self-signed certificate – This is a special case of self-issued certificates where the private key used by the CA to sign the certificate corresponds to the public key that is certified within the CA certificate. A CA might use a self-signed certificate, for example, to advertise their public key or other information about their operations.

– Cross-certificate – This is a CA certificate where the issuer and the subject are different CAs. CAs issue certificates to other CAs either as a mechanism to authorize the subject CA's existence (e.g., in a strict hierarchy) or to recognize the existence of the subject CA (e.g., in a distributed trust model). The cross-certificate structure is used for both of these.

7.5 Trust anchor

An entity is a trust anchor for a particular relying party for one or more purposes, typically including public-key certificate validation. A trust anchor is identified by trust anchor information. Trust anchor information includes a public key and some associated data. This trust anchor information is configured into the relying party in a trust anchor store. A relying party may have configured information about multiple trust anchors into one or more trust anchor stores.

A trust anchor may act as a CA that issues public-key certificates and certification revocation lists (CRLs) (see clause 7.10). The relying party may then use the trust anchor information for public-key certificate and CRL validation.

A trust anchor may also act as a PKI end entity by signing other types of information such as software packages, time stamps, responses to online certificate status protocol (OCSP) requests (see IETF RFC 6960), AVLs, etc.

A CA may be a trust anchor for some entities with respect to particular public-key certificates, but may otherwise be an ordinary CA.

NOTE 1 – As an example, entities within a company may trust all the public-key certificates issued by the company CA. This CA is then the trust anchor for these local relying parties with respect to local issued public-key certificates. However, it might not be a trust anchor with respect to public-key certificates issued outside the company. Likewise, relying parties outside the company may not consider the company CA as the trust anchor for any public-key certificates.

NOTE 2 – The term trust anchor is seen as synonymous with the term root-CA. In a strict hierarchy, the CA at the top of the hierarchy may be the root CA and it may also be a trust anchor. However, in more complex environments, it may not be possible to identify a root CA. Even when it is possible to identify a root CA, a relying party may not necessarily consider it a trust anchor. An intermediate CA may instead take that role. The term root CA is confusing and therefore not used by this Specification.

IETF RFC 5914 defines trust anchor information as a choice among three alternatives:

```
TrustAnchorChoice ::= CHOICE {
  certificate      Certificate,
  tbsCert      [1] EXPLICIT TBSCertificate,
  taInfo       [2] EXPLICIT TrustAnchorInfo }
```

The **certificate** alternative specifies a public-key certificate that can be either a self-signed certificate or a public-key certificate.

The **tbsCert** alternative specifies an unsigned public-key certificate as defined in clause 7.2.

NOTE 3 – This alternative is deprecated by this Specification and therefore not considered further.

The **taInfo** alternative specifies a special trust anchor information format defined by IETF RFC 5914.

If the trust anchor information is not used for signing public-key certificates, it shall be an end-entity public-key certificate.

7.6 Entity relationship

There may be several CAs between the trust anchor recognized by the relying party and an entity acting as an end entity. Each CA has issued one or more cross-certificates for the next CA on the path between the trust anchor and the end entity. The CA that issues a cross-certificate to another CA takes the role of intermediate CA. The CA that is the subject for a cross-certificate takes the role of subject CA. This is illustrated in Figure 2. The same CA may take both the roles of an intermediate CA and a subject CA.

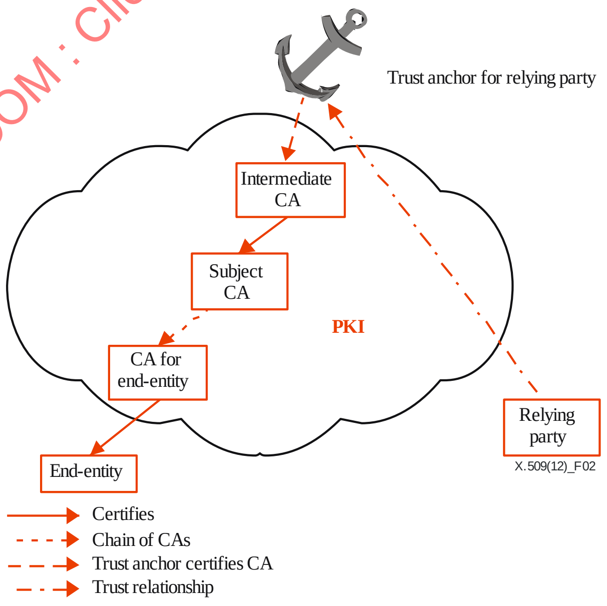

Figure 2 – Entity relationships

In some situations, conflicting or overlapping requirements for constraints, such as name constraints, may require a CA to issue more than one cross-certificate to another CA. In this case, multiple, different paths of public-certificate certificates are established between the trust anchor and the entity.

7.7 Certification path

Before a public-key certificate can be securely used by a relying party, it shall be validated. In order to validate such a public-key certificate, a chain of public-key certificates, called a certification path, shall be established between the public-key certificate signed by a trust anchor recognized by the relying party and the public-key certificate to be validated. Every public-key certificate within that path shall be checked. A certification path is thus an ordered list of public-key certificates starting with a public-key certificate signed by the trust anchor, and ending with the end-entity public-key certificate to be validated. All intermediate public-key certificates, if any, are CA certificates in which the subject of the preceding public-key certificate is the issuer of the following public-key certificate.

Each public-key certificate in a certification path shall be unique. A path that contains the same public-key certificate multiple times is not a valid certification path.

The **`issuer`** and **`subject`** components of each public-key certificate are used, in part, to identify a valid path. For each pair of adjacent public-key certificates in a valid certification path, the value of the **`subject`** component in one public-key certificate shall match the value of the **`issuer`** component in the subsequent public-key certificate. In addition, the value of the **`issuer`** component in the public-key certificate issued by the trust anchor shall match the distinguished name of the trust anchor. Only the names in these components are used when checking the validity of a certification path. Names in public-key certificate extensions are not used for this purpose. The **`distinguishedNameMatch`** matching rule, defined in clause 13.5.2 of Rec. ITU-T X.501 | ISO/IEC 9594-2, shall be used to compare the distinguished name in the **`subject`** component of one public-key certificate with the distinguished name in the **`issuer`** component of the subsequent public-key certificate.

Figure 3 illustrates the situation where a relying party needs to check the validity of an end-entity public-key certificate and the relying party is able to construct a certification path between one of its trust anchors and the end-entity public-key certificate.

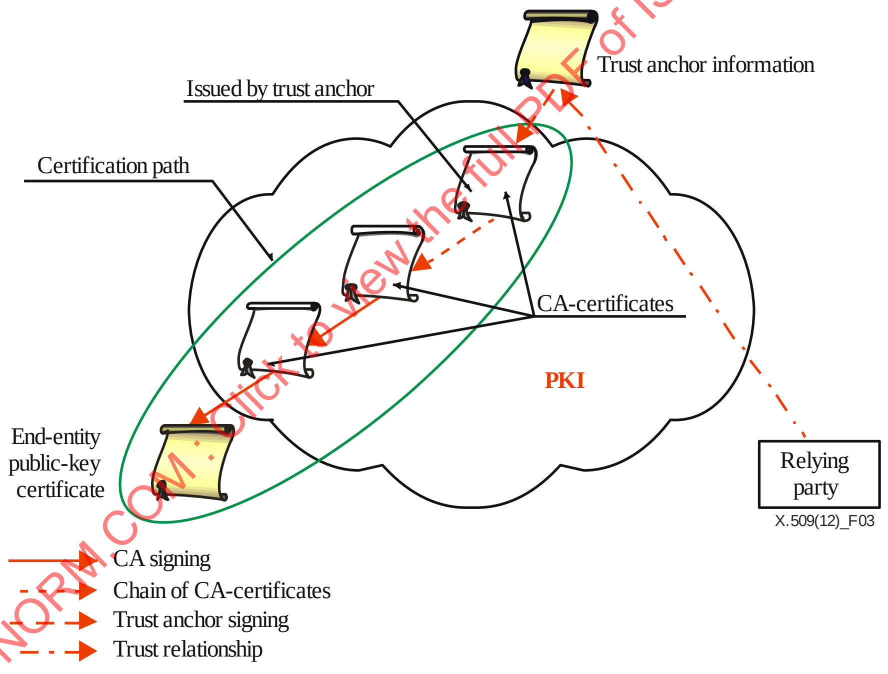

Figure 3 – Certification path

Trust suffers dilution as certification paths grow in length. The **`basicConstraints`** extension (see clause 9.4.2.1) allows restrictions to be put onto the length of the path. The validation of a public-key certificate may be affected by extensions in the chain of a public-key certificate, such as the **`certificatePolicies`** extension (see clause 9.2.2.6) and **`nameConstraints`** (see clause 9.4.2.2). It is the responsibility of the relying party to check that the restrictions are observed.

The ASN.1 data type **`PkiPath`** is used to represent a certification path. Within the sequence, the order of public-key certificates is such that the subject of the first public-key certificate is the issuer of the second public-key certificate, etc.

```
PkiPath ::= SEQUENCE OF Certificate
```

Each public-key certificate in a PKI path shall be unique.

The following data types are deprecated and should not be referenced by new specifications. They are retained here for backward compatibility not to invalidate existing specifications referencing these types.

NOTE – The **CertificationPath** data type was defined by the first edition of this Specification before the concept of certification path was fully developed. The order of elements in a **CertificationPath** instance is the opposite of that of a certification path. This data type is used, as an example, by the directory protocols for the support of strong authentication and digital signature (see Rec. ITU-T X.511 | ISO/IEC 9594-3). It is recommended that new applications use the **PkiPath** data type.

```
Certificates ::= SEQUENCE {
  userCertificate    Certificate,
  certificationPath  ForwardCertificationPath OPTIONAL,
  ... }
```

The following ASN.1 data type can be used to represent the forward certification path. This component contains the certification path that can point back to the originator.

```
ForwardCertificationPath ::= SEQUENCE OF CrossCertificates

CrossCertificates ::= SET OF Certificate

CertificationPath ::= SEQUENCE {
  userCertificate    Certificate,
  theCACertificates  SEQUENCE SIZE (1..MAX) OF CertificatePair OPTIONAL,
  ... }
```

The **userCertificate** component shall hold the end-entity public-key certificate.

The **theCACertificates** component may hold an element for each CA from the end entity up to and including the CA which has been certified by the trust anchor. If the end-entity public-key certificate has been issued directly by the trust anchor, this component shall be absent.

7.8 Generation of key pairs

The overall security management policy of an implementation shall define the lifecycle of key pairs, and is, thus, outside the scope of this Specification. However, it is vital to the overall security that all private keys remain known only to the entity (subject) to whom they belong.

Key data is not easy for a human user to remember, so a suitable method for storing it in a convenient transportable manner shall be employed. One possible mechanism when an entity is associated with a human user would be to use a "Smart Card". This would hold the private and (optionally) public keys of the user, the user's public-key certificate, and a copy of the CA's public key. The use of this card shall additionally be secured by, e.g., at least the use of a personal identification number (PIN), increasing the security of the system by requiring the user to possess the card and to know how to access it. In other environments, e.g., in a machine-to-machine environment (M2M), a hardware security module (HSM) may be used for storing critical data. The exact method chosen for storing such data, however, is beyond the scope of this Specification.

Three ways in which a key pair of an entity may be produced are:

a) The entity generates its own key pair. This method has the advantage that the private key of an entity is never released to another entity, but requires a certain level of competence by the entity.

b) The key pair is generated by a third party. The third party shall release the private key to the entity in a physically secure manner, and then actively destroy all information relating to the creation of the key pair plus the keys themselves. Suitable physical security measures shall be employed to ensure that the third party and the data operations are free from tampering.

c) The key pair is generated by the CA. This is a special case of b), and the considerations there apply.

NOTE – The CA already exhibits trusted functionality with respect to the entity, and shall be protected by the necessary physical security measures. This method has the advantage of not requiring secure data transfer to the CA for certification.

The cryptosystem in use imposes particular (technical) constraints on key generation.

7.9 Public-key certificate creation

A public-key certificate associates the public key and the unique name of the subject it describes. Thus:

a) a CA shall be satisfied of the identity of a subject before creating a public-key certificate for it;

b) a CA shall not issue public-key certificates for two different subjects with the same subject name.

It is important that the transfer of information to the CA is not compromised, and suitable physical security measures shall be taken. In this regard:

a) It would be a serious breach of security if the CA issued a public-key certificate for a subject with a public key that had been tampered with.

b) If the means of generation of key pairs of clause 7.8, item b) or item c) is employed, the subject's private key shall be transferred to the entity in a secure manner.

c) If the means of generation of key pairs of clause 7.8, item a) or item b) is employed, the subject may use different methods (online or offline) to communicate its public key to the CA in a secure manner. Online methods may provide some additional flexibility for remote operations performed between the entity and the CA.

A public-key certificate is a publicly available piece of information, and no specific security measures need to be employed with respect to its transmission e.g., to a directory server (directory system agent) according to the Directory Specifications or to the LDAP specification. As it is produced by an offline CA on behalf of a subject who shall be given a copy of it, the subject needs only store this information in its directory entry on a subsequent directory access. Alternatively, the CA could lodge the public-key certificate for the subject, in which case the CA shall be given suitable access rights to the entity's directory entry.

7.10 Certificate revocation list

7.10.1 Certificate revocation list principles

The CA that issues public-key certificates also has the responsibility to indicate the validity of the public-key certificates that it issues. Generally, public-key certificates are subject to possible subsequent revocation. This revocation and a notification of the revocation may be done directly by the same CA that issued the public-key certificate, or indirectly by another authority duly authorized by the CA that issued the public-key certificate. A CA that issues public-key certificates is required to state, possibly through a published statement of their practices, through the public-key certificates themselves, or through some other identified means, whether:

– the public-key certificates cannot be revoked;

– the public-key certificates may be revoked by the same CA directly; or

– the issuing CA authorizes a different entity to perform revocation.

CAs that do revoke public-key certificates are required to state, through similar means, what mechanism(s) can be used by relying parties to obtain revocation status information about public-key certificates issued by that CA. This Specification defines a certificate revocation list (CRL) mechanism and validation and authorization list (AVL) mechanism, but does not preclude the use of alternative mechanisms. One such alternative mechanism is the online certificate status protocol (OCSP) specified in IETF RFC 6960. Using this protocol, a relying party (client) requests the revocation status of a public-key certificate from an OCSP server. The server may use CRLs, or other mechanisms to check the status of the public-key certificate and respond to the client accordingly. If OCSP can be used by relying parties to check the status of a public-key certificate, IETF RFC 5280 contains a certificate extension (Authority Info Access) that would be included in such public-key certificates and would provide sufficient information to access an appropriate OCSP server. Relying parties check revocation status information, as appropriate, for all public-key certificates considered during the path processing procedure described in clause 12 to validate a public-key certificate.

Only a CA that is authorized to issue CRLs may choose to delegate that authority to another entity. If this delegation is done, it shall be verifiable at the time of public-key certificate/CRL verification. The **cRLDistributionPoints** extension (see clause 9.6.2.1) can be used for this purpose. The **cRLIssuer** component of this extension would be populated with the name(s) of any entities, other than the CA itself, that have been authorized to issue CRLs concerning the revocation status of the public-key certificate in question.

Public-key certificates shall have a lifetime associated with them, at the end of which they expire. In order to provide continuity of service, the authority shall ensure timely availability of replacement public-key certificates to supersede expired/expiring public-key certificates. Revocation notice date is the date/time that a revocation notice for a public-key certificate first appears on a CRL, regardless of whether it is a base or dCRL. In the CRL, revocation notice date is the value contained in the **thisUpdate** component. Revocation date is the date/time the CA actually revoked the public-key certificate, which could be different from the first time it appears on a CRL. In the CRL, revocation date is the value contained in the **revocationDate** component. Invalidity date is the date/time at which it is known or suspected that the private key was compromised or that the public-key certificate should otherwise be considered invalid. This date may be earlier that the revocation date. In the CRL, invalidity date is the value contained in the **invalidityDate** entry extension.

Two related points are:

– Validity of public-key certificates may be designed so that each becomes valid at the time of expiry of its predecessor, or an overlap may be allowed. The latter prevents the CA from having to install and distribute a large number of public-key certificates that may run out at the same expiration date.

– Expired public-key certificates will normally be removed from a directory. It is a matter for the security policy and responsibility of the CA to keep old public-key certificates for a period if a non-repudiation of data service is provided.

Public-key certificates may be revoked prior to their expiration time, e.g., if the entity's private key is assumed to be compromised, the entity is no longer to be certified by the CA, or if the CA's certificate is assumed to be compromised. The revocation of an end-entity public-key certificate or a CA certificate shall be made known by the issuing CA, and a new public-key certificate shall be made available, if appropriate. The CA may then inform the holder of the public-key certificate about its revocation by an offline procedure.

A CA that issues and subsequently revokes public-key certificates:

a) may be required to maintain an audit record of its revocation events for all public-key certificate types issued by that CA:

b) shall provide revocation status information to relying parties using CRLs, online certificate status protocol (OCSP), AVLs or another mechanism for the publication of revocation status information;

c) if using CRLs, it shall maintain and publish CRLs even if the lists of revoked public-key certificates are empty;

d) if using only partitioned CRLs, it shall issue a full set of partitioned CRLs covering the complete set of public-key certificates whose revocation status will be reported using the CRL mechanism. Thus, the complete set of partitioned CRLs shall be equivalent to a full CRL for the same set of public-key certificates, if the CRL issuer was not using partitioned CRLs.

Relying parties may use a number of mechanisms to locate revocation status information provided by an authority. For example, there may be a pointer in the public-key certificate itself that directs the relying party to a location where revocation information is provided. There may be a pointer in a revocation list that redirects the relying party to a different location. The relying party may locate revocation information in a repository (e.g., a directory) or through other means outside the scope of this Specification (e.g., locally configured).

If revocation lists are published in a directory, they are held within directory entries as attributes of the following types:

– **`certificateRevocationList`**;

– **`authorityRevocationList`**; and

– **`deltaRevocationList`**.

7.10.2 Certificate revocation list syntax

The following ASN.1 data type specifies the syntax of a CRL.

```
CertificateList ::= SIGNED{CertificateListContent}

CertificateListContent ::= SEQUENCE {
  version                Version OPTIONAL,
  -- if present, version shall be v2
  signature              AlgorithmIdentifier{{SupportedAlgorithms}},
  issuer                 Name,
  thisUpdate             Time,
  nextUpdate             Time OPTIONAL,
  revokedCertificates    SEQUENCE OF SEQUENCE {
    serialNumber           CertificateSerialNumber,
    revocationDate         Time,
    crlEntryExtensions     Extensions OPTIONAL,
    ...} OPTIONAL,
  ...,
  ...,
  crlExtensions     [0]  Extensions OPTIONAL }
```

The **`version`** component shall indicate the version of the encoded revocation list. If the **`extensions`** component is present in the revocation list, the version shall be **`v2`**. If the **`extensions`** component is not present, the version shall either be absent or be present as **`v2`**.

NOTE 1 – In the first and the second editions of this specification, the version component was always absent. In the third, fourth, fifth and sixth editions of this specification, the version shall be v2, if the extensions component flagged as critical is present in the revocation list. Or the version may either be absent or present as v2, if no extensions component flagged as critical is present in the revocation list.

The **`signature`** component shall contain the identifier for the algorithm used by the CA to sign the revocation list. It shall be the same value as used in the **`algorithmIdentifier`** component of the **`SIGNATURE`** data type when signing the revocation list.

NOTE 2 – By including this component, the signature algorithm is protected by the signature.

The **`issuer`** component shall identify the entity that signed and issued the revocation list.

The **`thisUpdate`** component shall indicate the date/time at which this revocation list was issued.

The **`nextUpdate`** component, when present, shall indicate the date/time by which the next revocation list in this series will be issued. The next revocation list could be issued before the indicated date, but it shall not be issued any later than the indicated time.

The **`revokedCertificates`** component shall identify public-key certificates that have been revoked. The revoked public-key certificates are identified by their serial numbers. If none of the public-key certificates covered by this CRL has been revoked, it is strongly recommended that the **`revokedCertificates`** parameter be omitted from the CRL, rather than being included with an empty **`SEQUENCE`**.

The **`crlExtensions`** component, when present, shall contain one or more CRL extensions.

NOTE 3 – The checking of the entire list of public-key certificates is a local matter. The list shall not be assumed to be in any particular order unless specific ordering rules have been specified by the issuing CA, e.g., in that CA's policy.

NOTE 4 – If a non-repudiation of data service is dependent on keys provided by the CA, the service should ensure that all relevant keys of the CA (revoked or expired) and the time stamped revocation lists are archived and certified by a current authority.

When an implementation processing a CRL encounters the serial number of the public-key certificate of interest in a CRL entry, but does not recognize a critical extension in the **`crlEntryExtensions`** component from that CRL entry, that CRL cannot be used to determine the status of the public-key certificate. When an implementation does not recognize a critical extension in the **`crlExtensions`** component, that CRL cannot be used to determine the status of the public-key certificate, regardless of whether the serial number of the public-key certificate of interest appears in that CRL or not.

NOTE 5 – In these cases, local policy may dictate actions in addition to and/or stronger than those stated in this Specification, such as seeking revocation status information from other sources.

Public-key certificates for which revocation status cannot be determined should not be considered valid public-key certificates.

If an extension affects the treatment of the list (e.g., multiple CRLs need to be scanned to examine the entire list of revoked public-key certificates, or an entry may represent a range of public-key certificates), then either that extension or a related extension shall be indicated as critical in the **`crlExtensions`** component. Therefore, a critical extension in the **`crlEntryExtensions`** component of an entry shall affect only the public-key certificate specified in that entry, unless there is a related critical extension in the **`crlExtensions`** component that advertises a special treatment for it. The only example of this situation defined in this Directory Specification is the **`certificateIssuer`** CRL entry extension and the related **`issuingDistributionPoint`** CRL extension when the **`indirectCRL`** Boolean from that extension is set to **`TRUE`**.

NOTE 6 – Extensions for CRLs are defined in clause 9 of this Specification.

If unknown components appear within an extension, and the extension is not flagged as critical, those unknown components shall be ignored according to the rules of extensibility documented in clause 12.2.2 of Rec. ITU-T X.519 | ISO/IEC 9594-5.

7.11 Uniqueness of names

A PKI requires that CAs are uniquely and unambiguously named. If CRL issuing authorities are not uniquely named, it may result in incorrect use of revocation information.

It is outside the scope of this Specification to specify procedures that ensure unique and unambiguous names for CA and CRL issuing authorities.

7.12 Indirect CRLs

7.12.1 Introduction

The only mechanism defined for CRL delegation by this Specification (and IETF RFC 5280) is for the public-key certificate issuing CA to include a **`cRLDistributionPoint`** extension in a public-key certificate and include the **`cRLIssuer`** component in this extension. The public-key certificate issuing CA will have do this for each public-key certificate whose revocation status the CA wishes to delegate via CRL to a CRL issuing authority.

There is no mechanism (i.e., public-key certificate or CRL extension) for a public-key certificate issuing CA to delegate CRL issuance for all its public-key certificate to another authority using a mechanism (similar to the delegated OCSP Responder public-key certificate as specified in IETF RFC 6960).

For example, if a CA has issued a large number of public-key certificates and its wishes to delegate CRL issuance for all of these public-key certificate to a CRL issuing authority, the CA shall assert the **`cRLIssuer`** component in the **`cRLDistributionPoint`** extension of each of the issued public-key certificates. If the CA wishes to delegate issuance only for some of the issued public-key certificates, the CA shall assert the **`cRLIssuer`** component in the **`cRLDistributionPoint`** extension of the delegated public-key certificates and shall not assert the **`cRLIssuer`** component in the remaining public-key certificates that contain the **`cRLDistributionPoint`** extension.

The relationship of CRL delegation may be as follows:

a) A CA can delegate issuance of CRL for a given public-key certificate to multiple CRL issuance authorities. The CA might delegate to multiple authorities for the sake of redundancy by asserting multiple CRL issuers in a single distribution point in the **`cRLDistributionPoint`** extension or by asserting multiple distribution points in the **`cRLDistributionPoint`** extension with each distribution point containing one or more CRL issuer(s). Another example is a CA delegating CRL issuance to different authorities for different reason codes. In this case, the CRL Distribution Point extension must contain two or more distribution points with each distribution point containing applicable reason code(s) and CRL Issuer(s).

b) A CA can delegate issuance of CRL for different batch of public-key certificates to different CRL issuance authorities. The CA could create these batches stochastically or using a deterministic algorithm such as based on type of public-key certificate, reason code, issuance time, expiration time, subject organization, etc.

c) A CRL issuance authority can be authoritative for revocation information for public-key certificates issued by multiple CAs.

7.12.2 Indirect CRL contents

If a CRL issuance authority is a CA, the CRLs it issues are authoritative for the public-key certificates issued by the CRL issuance authority as the CA and the public-key certificates whose revocation status is delegated to the CRL issuance authority. Thus, a CRL issued by a CRL issuance authority which has been delegated CRL issuance by *x* CAs, is authoritative for *x* + 1 or *x* CAs depending on whether the CRL issuance authority is a CA or not.

The CRL issued by the CRL issuance authority can be partitioned like any other CRLs using the **`distributionPoint`** component of the **`cRLDistributionPoint`** extension (see clause 9.6.2.1). Furthermore, the CRL issuance authority may or may not choose to partition the CRL based on the public-key certificate issuer. If it chooses the former, it creates a partitioned CRL for each CA. But, the partitioned CRL discussion is outside the scope of this Specification.

Since the iCRL is authoritative for the CA(s) other that the CRL issuer, serial number alone in the CRL entry does not uniquely identify a public-key certificate that has been revoked. You also need to identify the CA that issued the public-key certificate placed on the iCRL. This is achieved by adding the **`certificateIssuer`** CRL entry extension (see clause 9.6.2.3). This extension shall always be flagged as critical to ensure that the relying parties process it and associate the CRL entry with the appropriate CA.

If each entry on a CRL contained the **`certificateIssuer`** extension (which is a directory distinguished name), it would make the CRL size large. Thus, in order to reduce the CRL size, the iCRL issuing authority should sort the CRL entries by issuing CAs. Using this approach, only the first public-key certificate appearing on the CRL for a given CA needs to contain the **`certificateIssuer`** extension. All subsequent entries are assumed to be for the same public-key certificate issuing CA until another **`certificateIssuer`** CRL entry extension is encountered. To further reduce the size of the CRL, if the iCRL issuing authority is a CA, it should contain its revoked public-key certificate first, obviating the need for **`certificateIssuer`** extension for any of its certificates.

NOTE 1 – The following example illustrates the use of iCRLs. In the example, there is a single CRL issuing authority that issues revocation lists for multiple CAs. The **`issuingDistributionPoint`** extension (see clause 9.6.2.2) is present in that CRL and is flagged as critical. The **`indirectCRL`** component of this extension is set to **`TRUE`**. If the CRL issuing authority name is the same as that of the CAs it serves, entries should then be placed first on the CRL without the **`certificateIssuer`** CRL entry extension. Entries for other CAs are kept together and the first CRL entry for a particular CA includes the **`certificateIssuer`** CRL entry extension flagged as critical.

NOTE 2 – A relying party needs to develop and validate the certification path for the iCRL issuance authority. This is no different for building a certification path for a regular CRL with one difference. In the case of a regular CRL, there is a probability that the CRL is signed using the same key as the public-key certificate, obviating the need for building a CRL certification path. However, for the indirect CRL, the CRL certification path will always differ from the certification path for the public-key certificate whose revocation status is being checked.

7.13 Repudiation of a digital signing

Participants in an event may subsequently decide to repudiate anything that they digitally signed in that event. For example, one can dispute one's participation in a key establishment or being the originator of a signed email message as easily as one can dispute one's signing of a document with the intent to be bound to the content of that document. The repudiation may not be successful. The Non-repudiation Framework, Rec. ITU-T X.813 | ISO/IEC 10181-4, describes a dispute resolution process as follows:

1) evidence generation;
2) evidence transfer, storage and retrieval;
3) evidence verification; and
4) dispute resolution.

The generated evidence may include, but is not limited to:

– audit records pertinent to the event and an assertion of intent;
– statements made by third party notaries;
– policy statements;
– digitally signed information, including audit records and notary statements;
– timestamps of the digitally signed information;
– the public-key certificates supporting the digital signature;
– the appropriate revocation information published and available at the time of the disputed event; and,
– any public-key certificate revocations subsequent to the time of the event which indicate a key compromise occurred before the time of the event.

The integrity of stored data that might be presented as evidence may be maintained in a variety of ways, e.g., access control, storage of hashes by a trusted third party, digital signature. It may also be necessary periodically to strengthen the protection of that stored data to counteract improvements in computer processing and/or crypto-analysis.

NOTE – Neither the type and amount of evidence generated nor the level of integrity is specified by this Specification. However, it is expected that the level of effort will be commensurate with the risk involved.

Evidence verification may require the revalidation of the digital signatures of data, e.g., messages, documents, certificates, CRLs, and timestamps that were used in the initial validation process. The fact that a public-key certificate has expired shall not preclude its use for revalidating signatures created during the validity period of that public-key certificate. A public-key certificate that has been revoked may be used if it can be determined that the public-key certificate was valid at the time of the disputed event.

Even if all the digital evidence described above is considered technically valid, other conditions, e.g., the intent, understanding or competence of the signer, may allow the signer successfully to repudiate it.

8 Trust models

Procedures for determining the trustworthiness of a public-key certificate for a particular transaction are not defined here. In the three cornered trust model described below, the relying party acts on its own behalf to determine both the validity and trustworthiness of a public-key certificate. In the four-cornered trust model, the relying party engages the services of a trust broker to determine the trustworthiness of a public-key certificate on its behalf.

8.1 Three-cornered trust model

The three-cornered trust model applies to an environment where the entity to which a public-key certificate has been issued (the subject) and the relying party both have some relationship with the CAs within a closed environment.

This is illustrated by the simplified example shown in Figure 4, where the subject of the public-key certificate and the relying party are associated with the same CA. In this environment, the trustworthiness of the issuing CA is relatively easy to establish.

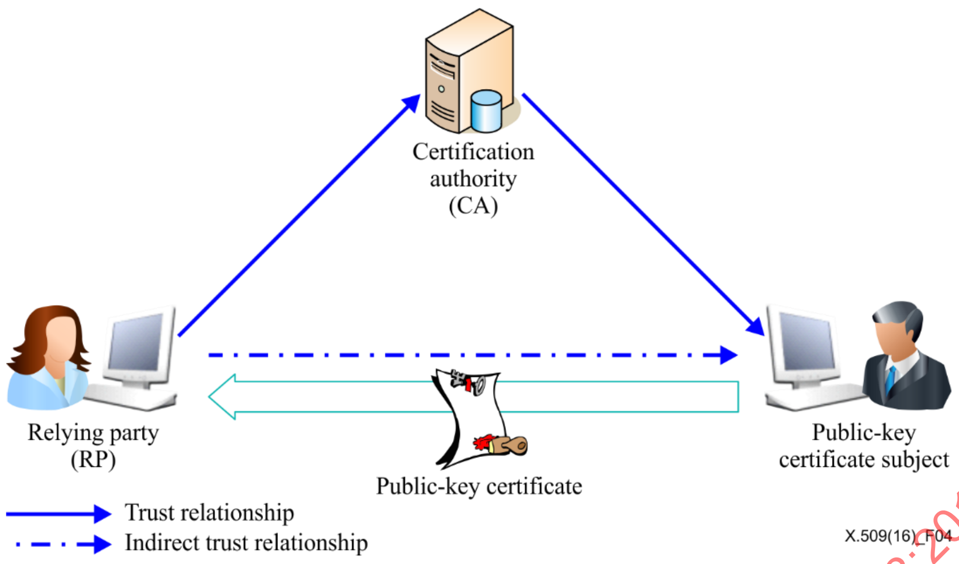
Certification authority (CA)
Relying party (RP)
Public-key certificate subject
Public-key certificate
Trust relationship
Indirect trust relationship
X.509(16)_F04

Figure 4 – The three cornered trust model

In this three cornered trust model, the public-key certificate subject trusts the CA and has asked it to issue a public-key certificate. Upon positive verification of the identity of the public-key certificate subject, the CA issues it with a public key certificate containing the subject's identifier. The relying party, being a public-key certificate subject of a CA within the same closed environment, also knows the trustworthiness of the issuing CA. Consequently the relying party can determine whether to trust the public-key certificate subject for the current transaction.

8.2 Four cornered trust model

The four cornered trust model may be used in open public-key infrastructures, where the relying party does not have a close relationship with the CA that issued the public-key certificate under validation. See Figure 5.

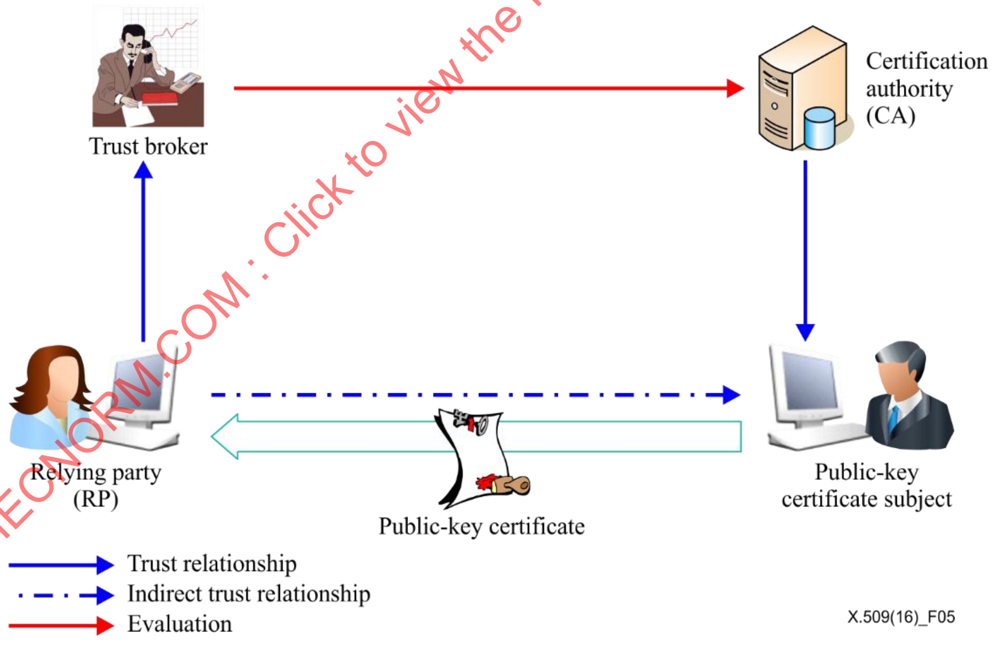

Figure 5 – The four cornered trust model

In the four cornered model, the relying party trusts an entity called the trust broker. The relying party is still responsible for validating public-key certificates and public-key certificate chains. However, the trust broker evaluates the trustworthiness of the CA for all intended transactions and relays this to the relying party so that it can decide about the trustworthiness of the public-key certificate subject for the current transaction. If the relying party's decision is positive, this means that the relying party trusts the CA and its public-key certificate for this transaction. The CA trusts the identity of the public-key certificate subject. Hence the relying party can indirectly trust the public-key certificate of the subject for this particular transaction.

9 Public-key certificate and CRL extensions

Some extensions defined in this clause are for use with public-key certificates. Other extensions are defined for CRLs (including CARLs and EPRLs). Extensions for use with attribute certificates and ACRLs (including AARLs and EARLs) are defined in clause 17.

This clause specifies extensions in the following areas:

a) Key and policy information: These public-key certificate and CRL extensions convey additional information about the keys involved, including key identifiers for subject and issuer keys, indicators of intended or restricted key usage, and indicators of a certificate policy.

b) Subject and issuer attributes: These public-key certificate and CRL extensions support alternative names, of various name forms, for a public-key certificate subject, a public-key certificate issuer, or a CRL issuer. These extensions can also convey additional attribute information about the public-key certificate subject, to assist a relying party in being confident that the public-key certificate subject is a particular person or entity.

c) Certification path constraints: These public-key certificate extensions allow constraint specifications to be included in CA certificates, i.e., CA certificates for CAs issued by other CAs, to facilitate the automated processing of certification paths when multiple certificate policies are involved. Multiple certificate policies arise when policies vary for different applications in an environment or when interoperation with external environments occurs. The constraints may restrict the types of public-key certificates that can be issued by the subject CA or that may occur subsequently in a certification path.

d) Basic CRL extensions: These CRL extensions allow a CRL to include indications of revocation reason, to provide for temporary suspension of a public-key certificate, and to include CRL-issue sequence numbers to allow relying parties to detect missing CRLs in a sequence from one CRL issuer.

e) CRL distribution points and delta CRLs: These public-key certificate and CRL extensions allow the complete set of revocation information from one CA to be partitioned into separate CRLs and allow revocation information from multiple CAs to be combined in one CRL. These extensions also support the use of partial CRLs indicating only changes since an earlier CRL issue.

Inclusion of any extension in a public-key certificate or CRL is at the option of the CA issuing that public-key certificate or the issuing authority for that CRL.

In a public-key certificate or CRL, an extension is flagged as being either critical or as non-critical. If an extension is flagged as critical and a relying party does not recognize the extension type or does not implement the semantics of the extension, then that relying party shall consider the public-key certificate as invalid. If an extension is flagged as non-critical, a relying party that does not recognize or implement that extension type may process the remainder of the public-key certificate ignoring the extension. If an extension is flagged as non-critical, a relying party that does recognize the extension, shall process the extension. Extension type definitions in this Specification indicate whether the extension shall always be flagged as critical, always be flagged as non-critical, or whether criticality can be decided by the public-key certificate or CRL issuer. The reason for requiring some extensions to be always flagged as non-critical is to allow relying parties which do not need to use such extensions to omit support for them without jeopardizing the ability to interoperate with all CAs.

NOTE – A relying party may require certain non-critical extensions to be present in a public-key certificate in order for that public-key certificate to be considered acceptable. The need for inclusion of such extensions may be implied by local policy rules of the relying party or may be a CA policy rule indicated to the relying party by inclusion of a particular public-key certificate policy identifier in the public-key certificate policies extension with that extension being flagged as critical.

For all public-key certificate extensions, CRL extensions, and CRL entry extensions defined in this Specification, there shall be no more than one instance of each extension type in any public-key certificate, CRL, or CRL entry, respectively.

9.1 Policy handling

9.1.1 Certificate policy

This framework contains four types of entities: The relying party, the CA, the trust broker, and the public-key certificate subject. Each entity operates under obligations to the other entities and, in return, enjoys limited warranties offered by them. The obligations and warranties of a CA are defined in its certificate policy. A certificate policy is a document (usually in plain-language, but it could be machine readable). It may be referenced by an object identifier and a URL, which may be included in the certificate policies extension of the public-key certificate issued by the CA to the entity and upon which the relying party relies. A public-key certificate may be issued in accordance with one or more policies. The definition of the policy and assignment of the object identifier is performed by a policy authority. The set of policies administered by a policy authority is called a policy domain. All public-key certificates are issued in accordance with a policy, even if the policy is neither recorded anywhere nor referenced in the public-key certificate. This Specification does not prescribe the style or contents of the certificate policy.

The relying party may be bound to its obligations under the certificate policy by the act of importing a CA public key and using it as trust anchor information, or by relying on a public-key certificate that includes the associated policy identifier. The CA may be bound to its obligations under the policy by the act of issuing a public-key certificate that includes the associated policy identifier. The public-key certificate subject and any relying party who is also a public-key certificate subject may be bound to its obligations under the policy by the act of requesting and accepting a public-key certificate that includes the associated policy identifier and by using the corresponding private key. Implementations that do not use the certificate policies extension should achieve the required binding by other means.

The relying party and trust broker may be bound by any contractual agreement that they have or by any trust policy that the trust broker issues.

For an entity simply to declare conformance to a policy does not generally satisfy the assurance requirements of the other entities in the framework. They require some reason to believe that the other parties operate a reliable implementation of their policy. However, if explicitly stated in the certificate policy, relying parties may accept the CA's assurances that its subjects agree to be bound by their obligations under the policy, without having to confirm this directly with them. This aspect of certificate policy is outside the scope of this Specification.

A CA may place limitations on the use of its public-key certificates, in order to control the risk that it assumes as a result of issuing public-key certificates. For instance, it may restrict the community of relying parties, the purposes for which they may use its public-key certificates and/or the type and extent of damages that it is prepared to make good in the event of a failure on its part, or that of its entities acting as PKI end entities. These matters should be defined in the certificate policy.

Additional information, to help affected entities understand the provisions of the certificate policy, may be included in the certificate policies extension in the form of policy qualifiers.

9.1.2 Cross-certificates and policy handling

The warranties and obligations shared by the subject CA, the intermediate CA and the relying party are defined by the certificate policy identified in the cross-certificate, in accordance with which the subject CA may act as, or on behalf of, an end entity. The warranties and obligations shared by the public-key certificate subject, the subject CA and the intermediate CA are defined by the certificate policy identified in the end-entity public-key certificate, in accordance with which the intermediate CA may act as, or on behalf of, a relying party.

A certification path is said to be valid under the set of certificate policies that are common to all public-key certificates in the path.

In addition to the situation described above, there are two special cases to be considered:

a) the CA does not use the certificate policies extension to convey its policy requirements to relying parties; and

b) the relying party or intermediate CA delegates the job of controlling policy to the next CA in the path.

In the first case, the public-key certificate should not contain a certificate policies extension at all. As a result, the set of policies under which the path is valid will be null. But, the path may be valid nonetheless. Relying parties shall still ensure that they are using the public-key certificate in conformance with the policies of the CAs in the path.

In the second case, the relying party or intermediate CA should include the special value *any-policy* in the *initial-policy-set* or cross-certificate. Where a public-key certificate includes the special value *any-policy*, it should not include any other certificate policy identifiers. The identifier *any-policy* should not have any associated policy qualifiers.

The relying party can ensure that all its obligations are conveyed by setting the *initial-explicit-policy* indicator. In this way, only authorities that use the certificate policies extension as their way of achieving binding are accepted in the path, and relying parties have no additional obligations. Because CAs also attract obligations when they act as, or on behalf of, a relying party, they can ensure that all their obligations are conveyed by setting the **`requireExplicitPolicy`** component of the **`policyConstraints`** extension in the cross-certificate.

9.1.3 Policy mapping

Some certification paths may cross boundaries between policy domains. The warranties and obligations according to which the cross-certificate is issued may be materially equivalent to some or all of the warranties and obligations according to which the subject CA issues end-entity public-key certificates, even though the policy authorities under which the two CAs operate may have selected different object identifiers for these materially equivalent policies. In this case, the intermediate CA may include a policy mappings extension in the cross-certificate. In the policy mappings extension, the intermediate CA assures the relying party that it will continue to enjoy the familiar warranties, and that it should continue to fulfil its familiar obligations, even though subsequent entities in the certification path operate in a different policy domain. The intermediate CA should include one or more mappings for each of a subset of the policies under which it issued the cross-certificate, and it should not include mappings for any other policies. If one or more of the certificate policies according to which the subject CA operates is identical to those according to which the intermediate CA operates (i.e., it has the same unique identifier), then these identifiers should be excluded from the policy mapping extension, but included in the certificate policies extension.

Policy mapping has the effect of converting all policy identifiers in public-key certificates further down the certification path to the identifier of the equivalent policy, as recognized by the relying party.

Policies shall not be mapped either to or from the special value *any-policy*.

Relying parties may determine that public-key certificates issued in a policy domain other than its own should not be relied upon, even though a trusted intermediate CA may determine its policy to be materially equivalent to its own. It can do this by setting the *initial-policy-mapping-inhibit input* to the path validation procedure. Additionally, an intermediate CA may make a similar determination on behalf of its relying parties. In order to ensure that relying parties correctly enforce this requirement, it can set **`inhibitPolicyMapping`** in a **`policyConstraints`** extension.

9.1.4 Certification path processing

The relying party faces a choice between two strategies:

a) it can require that the certification path be valid under at least one of a set of policies pre-determined by the relying party; or

b) it can ask the path validation module to report the set of policies for which the certification path is valid.

The first strategy may be most appropriate when the relying party knows, *a priori*, the set of policies that are acceptable for its intended use.

The second strategy may be the most appropriate when the relying party does not know, *a priori*, the set of policies that are acceptable for its intended use.

In the first instance, the certification path validation procedure will indicate the path to be valid only if it is valid under one or more of the policies specified in the *initial-policy-set*, and it will return the sub-set of the *initial-policy-set* under which the path is valid. In the second instance, the certification path validation procedure may indicate that the path is invalid under the *initial-policy-set*, but valid under a disjoint set: the *authorities-constrained-policy-set*. Then the relying party shall determine whether its intended use of the public-key certificate is consistent with one or more of the certificate policies under which the path is valid. By setting the *initial-policy-set* to *any-policy*, the relying party can cause the procedure to return a valid result if the path is valid under any (unspecified) policy.

9.1.5 Self-issued certificates

A CA may issue a certificate to itself under three circumstances:

a) as a convenient way of encoding the public key associated with the private key used to sign the public-key certificate, so that it can be communicated to and stored as trust anchor information by relying parties;

b) for certifying additional public keys of the CA used for purposes other than those covered by category a) (such as OCSP and possibly CRL signing); and

c) for replacing its own expired CA certificates.

These types of CA certificates are called self-issued certificates, and they can be recognized by the fact that the issuer and subject names present in them are identical. For the purposes of path validation, self-issued certificates of category a) are self-signed certificates and are therefore verified with the public key contained in them, and if they are encountered in the path, they shall be ignored.

Self-issued certificates of type b) may only appear as end-entity public-key certificates in a path, and shall be processed as end-entity public-key certificates.

Self-issued certificates of type c) (also known as self-issued intermediate certificates) may appear as intermediate certificates in a path. As a matter of good practice, when replacing a key that is on the point of expiration, a CA should request the issuance of any in-bound cross-certificates that it requires for its replacement public key before using the key. Nevertheless, if self-issued certificates of this category are encountered in the path, they shall be processed as intermediate certificates, with the following exception: they do not contribute to the path length for the purposes of processing the **`pathLenConstraint`** component of the **`basicConstraints`** extension and the *skip-certificates* values associated with the *policy-mapping-inhibit-pending* and *explicit-policy-pending indicators*.

If a CA uses the same key to sign public-key certificates and CRLs, a single self-issued certificate of category a) shall be used. If a CA uses a different key to sign CRLs than that used to sign public-key certificates, the CA may choose to issue two self-issued public-key certificates of category a), one for each of the keys. In this situation, relying parties would need access to both self-issued public-key certificates to establish separate trust anchors for public-key certificates and CRLs signed by that CA. Alternatively, a CA may issue one self-issued public-key certificate of category a) for public-key certificate signing and one self-issued public-key certificate of category b) for CRL signing. In this situation, relying parties use the key certified in the public-key certificate of category a) as their single trust anchor for both public-key certificates and CRLs signed by that CA. In this case, if the self-issued public-key certificate of category b) were to be used to verify signatures on CRLs, there is no means defined in this Specification to check the validity of that public-key certificate.

If self-issued certificates of category b) are encountered within a path, they shall be ignored.

NOTE – Other mechanisms for distributing CA public keys are outside the scope of this Specification.

9.2 Key and policy information extensions

9.2.1 Requirements

The following requirements relate to key and policy information:

a) CA key pair updating can occur at regular intervals or in special circumstances. There is a need for a certificate extension to convey an identifier of the public key to be used to verify the public-key certificate signature. A relying party can use such identifiers in finding the correct CA certificate for validating the public-key certificate issuer's public key.

b) In general, a public-key certificate subject has different public keys and, correspondingly, different public-key certificates for different purposes, e.g., digital signature and encipherment key agreement. A public-key certificate extension is needed to assist a relying party in selecting the correct public-key certificate for a given subject for a particular purpose or to allow a CA to stipulate that a certified key may only be used for a particular purpose.

c) Subject key pair updating can occur at regular intervals or in special circumstances. There is a need for a public-key certificate extension to convey an identifier to distinguish between different public keys for the same subject used at different points in time. A relying party can use such identifiers in finding the correct public-key certificate.

d) The private key corresponding to a certified public key is typically used over a different period from the validity of the public key. With a key pair used for creating and validating signatures, the usage period for the private key used for signing is typically shorter than that for the public key used to verify the signature. The validity period of the public-key certificate indicates a period for which the public key may be used, which is not necessarily the same as the usage period of the private key. In the event of a private key compromise, the period of exposure can be limited if the relying party knows the legitimate use period for the private key. There is therefore a requirement to be able to indicate the usage period of the private key in a public-key certificate.

e) Because public-key certificates may be used in environments where multiple certificate policies apply, provision needs to be made for including certificate policy information in public-key certificates.

f) When cross-certifying from one organization to another, it can sometimes be agreed that certain of the two organizations' policies can be considered equivalent. A CA certificate needs to allow the issuing CA to indicate that one of its own certificate policies is equivalent to another certificate policy in the subject CA's domain. This is known as policy mapping.

g) A user of an encipherment or digital signature system which uses public-key certificates defined in this Specification needs to be able to determine in advance the algorithms supported by other users.

h) There is a need to specify that an end-entity public-key certificate shall be validated using an authorization and validation list.

9.2.2 Public-key certificate and CRL extensions

The following extensions are defined:

a) authority key identifier;

b) subject key identifier;

c) key usage;

d) extended key usage;

e) private key usage period;

f) certificate policies;

g) policy mappings

h) authorization and validation.

These extensions shall be used only as public-key certificate extensions, except for the **`authorityKeyIdentifier`** extension which may also be used as a CRL extension. Unless otherwise noted, these extensions may be used in both CA certificates and end-entity public-key certificates.

9.2.2.1 Authority key identifier extension

This extension, which may be used as either a public-key certificate extension or CRL extension, identifies the public key to be used to verify the signature on this public-key certificate or CRL. It enables distinct keys used by the same CA to be distinguished (e.g., as key updating occurs) and it enables distinct keys used by the same CRL issuer to be distinguished. This extension is defined as follows:

```
authorityKeyIdentifier EXTENSION ::= {
  SYNTAX          AuthorityKeyIdentifier
  IDENTIFIED BY   id-ce-authorityKeyIdentifier }

AuthorityKeyIdentifier ::= SEQUENCE {
  keyIdentifier               [0]  KeyIdentifier OPTIONAL,
  authorityCertIssuer         [1]  GeneralNames OPTIONAL,
  authorityCertSerialNumber   [2]  CertificateSerialNumber OPTIONAL,
  ... }
  (WITH COMPONENTS {..., authorityCertIssuer        PRESENT,
                         authorityCertSerialNumber  PRESENT } |
   WITH COMPONENTS {..., authorityCertIssuer        ABSENT,
                         authorityCertSerialNumber  ABSENT } )

KeyIdentifier ::= OCTET STRING
```

The key may be identified by an explicit key identifier in the **keyIdentifier** component, by the identification of a public-key certificate for the key (giving certificate issuer in the **authorityCertIssuer** component and public-key certificate serial number in the **authorityCertSerialNumber** component), or by both explicit key identifier and identification of a public-key certificate for the key. If both forms of identification are used then the public-key certificate or CRL issuer shall ensure they are consistent. A key identifier shall be unique with respect to all key identifiers for the issuing authority for the public-key certificate or CRL containing the extension. An implementation which supports this extension is not required to be able to process all name forms in the **authorityCertIssuer** component (see clause 9.3.2.1 for details of the **GeneralNames** data type).

CAs shall assign public-key certificate serial numbers such that every (issuer, public-key certificate serial number) pair uniquely identifies a single public-key certificate. The **keyIdentifier** component can be used to select CA certificates during path construction. The **authorityCertIssuer**, **authoritySerialNumber** pair can only be used to provide preference to one public-key certificate over others during path construction.

This extension shall always be flagged as non-critical.

9.2.2.2 Subject key identifier extension

This extension identifies the public key being certified. It enables distinct keys used by the same subject to be differentiated (e.g., as key updating occurs). This extension is defined as follows:

```
subjectKeyIdentifier EXTENSION ::= {
  SYNTAX          SubjectKeyIdentifier
  IDENTIFIED BY   id-ce-subjectKeyIdentifier }

SubjectKeyIdentifier ::= KeyIdentifier
```

A key identifier shall be unique with respect to all key identifiers for the subject with which it is used.

This extension shall always be flagged as non-critical.

9.2.2.3 Key usage extension

This extension identifies the intended usage for which the public-key certificate has been issued. The intended usage may be further constrained by policy. This policy may be stated in a certificate policy definition, a contract, or other specification. However, a policy shall not override the constraint indicated by a **KeyUsage** bit, e.g., a certificate policy could not allow a public-key certificate to be used for digital signature if **KeyUsage** indicated that it could only be used for key agreement.

Setting a specific value of **KeyUsage** in a public-key certificate does not in itself signal for an instance of communication that the communicating parties are acting in accordance with this setting, e.g., when signing a document. The definition of methods by which parties may signal their intent for a specific instance of communication (e.g., commitment to content for that specific instance) is outside the scope of this Specification, but it is anticipated that multiple methods will exist. Although not recommended, it is possible to use the content of the public-key certificate, e.g., certificate policy, to signal the intent of the signing. However, since that signal was made when the public-key certificate was issued by the CA, such use may not meet the requirement that declaring the intent is made at the time of signing by the signer.

More than one bit may be set in an instance of the **keyUsage** extension. The setting of multiple bits shall not change the meaning of each individual bit but shall indicate that the public-key certificate may be used for all of the purposes indicated by the set bits. There may be risks incurred when setting multiple bits. A review of those risks is documented in Annex K.

This extension is defined as follows:

```
keyUsage EXTENSION ::= {
  SYNTAX          KeyUsage
  IDENTIFIED BY  id-ce-keyUsage }

KeyUsage ::= BIT STRING {
  digitalSignature   (0),
  contentCommitment (1),
  keyEncipherment    (2),
  dataEncipherment   (3),
  keyAgreement       (4),
  keyCertSign        (5),
  cRLSign            (6),
  encipherOnly       (7),
  decipherOnly       (8)  }
```

Bits in the **KeyUsage** type are as follows:

a) **digitalSignature**: for verifying digital signatures that are used with an entity authentication service, a data origin authentication service and/or an integrity service;

b) **contentCommitment**: for verifying digital signatures which are intended to signal that the signer is committing to the content being signed. The type of commitment the public-key certificate can be used to support may be further constrained by the CA, e.g., through a certificate policy. The precise type of commitment of the signer e.g., "reviewed and approved" or "with the intent to be bound", may be signalled by the content being signed, e.g., the signed document itself or some additional signed information.

   Since a content commitment signing is considered a digitally signed transaction, the **digitalSignature** bit need not be set in the certificate. If it is set, it does not affect the level of commitment the signer has endowed in the signed content.

   It is not incorrect to refer to this **keyUsage** bit using the identifier **nonRepudiation**. However, the use of this identifier has been deprecated. Regardless of the identifier used, the semantics of this bit are as specified in this Specification;

c) **keyEncipherment**: for enciphering keys or other security information, e.g., for key transport;

d) **dataEncipherment**: for enciphering user data, but not keys or other security information as in c);

e) **keyAgreement**: for use as a public key agreement key, such an when an asymmetric Diffie-Hellman key pair is used for key management.

f) **keyCertSign**: for verifying a CA's signature on public-key certificates.

   Since public-key certificate signing is considered a commitment to the content of the public-key certificate by the CA, neither the **digitalSignature** bit nor the **contentCommitment** bit need be set in the CA certificate. If either (or both) is set, it does not affect the level of commitment the signer has endowed in the signed public-key certificate;

g) **cRLSign**: for verifying an authority's signature on CRLs.

   Since CRL signing is considered to be commitment to the content of the CRL by the CRL issuer, neither the **digitalSignature** bit nor the **contentCommitment** bit need be set in the public-key certificate. If either (or both) is set, it does not affect the level of commitment the signer has endowed in the signed CRL;

h) **encipherOnly**: public key agreement key for use only in enciphering data when used with **keyAgreement** bit also set (meaning with other key usage bit set is undefined);

i) **decipherOnly**: public key agreement key for use only in deciphering data when used with **keyAgreement** bit also set (meaning with other key usage bit set is undefined).

Application specifications should indicate which of the **digitalSignature** or **contentCommitment** bits are appropriate for their use. If a signing application has no knowledge of the signer's intent regarding commitment to content, the application shall sign and support that signing with a public-key certificate that has the **digitalSignature** bit set in that public-key certificate's **keyUsage** extension.

Even though a digital signature was verified using a public-key certificate that has only the **digitalSignature** bit set, other factors external to the verification of the digital signature may also play a role in determining the intent of the signing.

Conversely, even though a digital signature was verified using a public-key certificate that has only the **`contentCommitment`** bit set, external factors may be used by the signer to disclaim commitment to the signed content.

The bit **`keyCertSign`** is for use in CA certificates only. If **`KeyUsage`** is set to **`keyCertSign`**, the value of the **`cA`** component of the **`basicConstraints`** extension shall be set to **`TRUE`**. CAs may also use other defined key usage bits in **`KeyUsage`**, e.g., **`digitalSignature`** for providing authentication and integrity of online administration transactions.

This extension may, at the option of the issuing CA be flagged as either critical or as non-critical.

If the extension is flagged as critical or if the extension is flagged as non-critical but the relying party recognizes it, then the public-key certificate shall be used only for a purpose for which the corresponding key usage bit is set to one. If the extension is flagged as non-critical and the relying party does not recognize it, then this extension shall be ignored.

A bit set to zero indicates that the key is not intended for that purpose. If the extension is present with all bits set to zero, the key is intended for a purpose other than those listed above.

9.2.2.4 Extended key usage extension

This extension indicates one or more purposes for which the certified public key may be used, in addition to, or in place of the basic purposes indicated in the key usage extension. This extension is defined as follows:

```
extKeyUsage EXTENSION ::= {
  SYNTAX          SEQUENCE SIZE (1..MAX) OF KeyPurposeId
  IDENTIFIED BY   id-ce-extKeyUsage }

KeyPurposeId ::= OBJECT IDENTIFIER
```

A CA may assert any-extended-key-usage by using the **`anyExtendedKeyUsage`** object identifier. This enables a CA to issue a public-key certificate that contains **`KeyPurposeId`** object identifiers for extended key usages that may be required by relying parties, without restricting the public-key certificate to only those key usages. If extended key usage would restrict key usage, then the inclusion of this object identifier removes that restriction.

```
anyExtendedKeyUsage OBJECT IDENTIFIER ::= { id-ce-extKeyUsage  0 }
```

Key purposes may be defined by any organization with a need. Object identifiers used to identify key purposes shall be assigned in accordance with Rec. ITU-T X.660 | ISO/IEC 9834-1.

This extension may, at the option of the issuing CA, be flagged either as critical or as non-critical.

If the extension is flagged as critical, then the public-key certificate shall be used only for one of the purposes indicated.

If the extension is flagged as non-critical, then it indicates the intended purpose or purposes of the key, and may be used in finding the correct key/public-key certificate of an entity that has multiple keys/public-key certificates. If this extension is present, and the relying party recognizes and processes the **`extendedKeyUsage`** extension type, then the relying party shall ensure that the public-key certificate shall be used only for one of the purposes indicated. (A relying party may nevertheless require that a particular purpose be indicated in order for the public-key certificate to be acceptable.)

If a public-key certificate contains both a critical key usage extension and a critical extended key usage extension, then both extensions shall be processed independently and the public-key certificate shall only be used for a purpose consistent with both extensions. If there is no purpose consistent with both extensions, then the public-key certificate shall not be used for any purpose.

This Specification defines the following key purposes that can be included in the extended key usage extension:

```
keyPurposes OBJECT IDENTIFIER ::= {id-kp 1}

id-avlSign  OBJECT IDENTIFIER ::= {id-kp 2}
```

The id-**`avlSign`** key purpose is used to indicate that the private key may be used for signing an AVL.

Other purposes that can also be included are defined in other specifications, such as IETF RFC 5280.

9.2.2.5 Private key usage period extension

This extension indicates the period of use of the private key corresponding to the certified public key. It is applicable only for private key used for creating digital signatures. This extension is defined as follows:

```
privateKeyUsagePeriod EXTENSION ::= {
  SYNTAX          PrivateKeyUsagePeriod
  IDENTIFIED BY   id-ce-privateKeyUsagePeriod }
```

```
PrivateKeyUsagePeriod ::= SEQUENCE {
  notBefore  [0]  GeneralizedTime OPTIONAL,
  notAfter   [1]  GeneralizedTime OPTIONAL,
  ... }
  (WITH COMPONENTS {..., notBefore  PRESENT } |
   WITH COMPONENTS {..., notAfter   PRESENT } )
```

The **notBefore** component indicates the earliest date and time at which the private key may be used for signing. If the **notBefore** component is not present, then no information is provided as to when the period of valid use of the private key commences beyond that specified in the validity component of the public-key certificate. The **notAfter** component indicates the latest date and time at which the private key may be used for signing. If the **notAfter** component is not present then no information is provided as to when the period of valid use of the private key concludes beyond that specified in the validity component of the public-key certificate.

This extension shall always be flagged as non-critical.

NOTE 1 – The period of valid use of the private key may be different from the certified validity of the public key as indicated by the public-key certificate validity period. The usage period for the private key used for signing is typically shorter than that for the public key used for verifying the signature.

NOTE 2 – The period of use of the private key corresponding to a public key can only be enforced if both the private key and the corresponding public-key certificate are placed in a tamper-resistant hardware module that contains a reliable clock synchronized with UTC. When this is not the case, a signer may avoid using a signing private key up to the very end of the validity period of the public-key certificate. This is one possible use of this extension.

NOTE 3 – In general, this Specification does not associate any semantic with this extension. Any particular use of this extension will have to specify the semantic associated with that usage.

9.2.2.6 Certificate policies extension

This extension lists certificate policies, recognized by the issuing CA, that apply to the public-key certificate, together with optional qualifier information pertaining to these certificate policies. The list of certificate policies is used in determining the validity of a certification path, as described in clause 12. The optional qualifiers are not used in the certification path processing procedure, but relevant qualifiers are provided as an output of that process to the relying party to assist in determining whether a valid path is appropriate for the particular transaction. Typically, different certificate policies will relate to different applications which may use the certified key. The presence of this extension in an end-entity public-key certificate indicates the certificate policies for which this public-key certificate is valid. The presence of this extension in a CA certificate (cross-certificate) issued by one CA to another CA indicates the certificate policies for which certification paths containing this public-key certificate may be valid. This extension is defined as follows:

```
certificatePolicies EXTENSION ::= {
  SYNTAX         CertificatePoliciesSyntax
  IDENTIFIED BY  id-ce-certificatePolicies }

CertificatePoliciesSyntax ::= SEQUENCE SIZE (1..MAX) OF PolicyInformation

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

CertPolicyId ::= OBJECT IDENTIFIER

PolicyQualifierInfo ::= SEQUENCE {
  policyQualifierId  CERT-POLICY-QUALIFIER.&id({SupportedPolicyQualifiers}),
  qualifier          CERT-POLICY-QUALIFIER.&Qualifier
              ({SupportedPolicyQualifiers}{@policyQualifierId}) OPTIONAL,
  ... }

SupportedPolicyQualifiers CERT-POLICY-QUALIFIER ::= {...}
```

A value of the **PolicyInformation** data type identifies and conveys qualifier information for one certificate policy. The **policyIdentifier** component contains an identifier of a certificate policy and the **policyQualifiers** component contains policy qualifier values for that component.

This extension may, at the option of the issuing CA, be either flagged as critical or as non-critical.

If the extension is flagged as critical, it indicates that the public-key certificate shall only be used for the purpose, and in accordance with the rules implied by one of the indicated certificate policies. The rules of a particular policy may require the relying party to process the qualifier value in a particular way.

If the extension is flagged as non-critical, use of this extension does not necessarily constrain use of the public-key certificate to the policies listed. However, a relying party may require a particular policy to be present in order to use the public-key certificate (see clause 12). Policy qualifiers may, at the option of the relying party, be processed or ignored.

Certificate policies and certificate policy qualifier types may be defined by any organization with a need. Object identifiers used to identify certificate policies and certificate policy qualifier types shall be assigned in accordance with Rec. ITU-T X.660 | ISO/IEC 9834-1. A CA may assert any-policy by using the **anyPolicy** object identifier in order to trust a public-key certificate for all possible policies. Because of the need for identification of this special value to apply regardless of the application or environment, that object identifier is assigned in this Specification. No object identifiers will be assigned in this Specification for specific certificate policies. That assignment is the responsibility of the entity that defines the certificate policy.

```
anyPolicy OBJECT IDENTIFIER ::= {id-ce-certificatePolicies 0}
```

The identifier **anyPolicy** should not have any associated policy qualifiers.

The following ASN.1 information object class is used in defining certificate policy qualifier types:

```
CERT-POLICY-QUALIFIER ::= CLASS {
  &id                    OBJECT IDENTIFIER UNIQUE,
  &Qualifier             OPTIONAL }
WITH SYNTAX {
  POLICY-QUALIFIER-ID  &id
  [QUALIFIER-TYPE      &Qualifier] }
```

A definition of a policy qualifier type shall include:

– a statement of the semantics of the possible values; and

– an indication of whether the qualifier identifier may appear in a certificate policies extension without an accompanying value and, if so, the implied semantics in such a case.

NOTE – A qualifier may be specified as having any ASN.1 type. When the qualifier is anticipated to be used primarily with applications that do not have ASN.1 decoding functions, it is recommended that the type **OCTET STRING** be specified. The ASN.1 **OCTET STRING** value can then convey a qualifier value encoded according to any convention specified by the policy element defining organization.

9.2.2.7 Policy mappings extension

This extension, which shall be used in CA certificates only, allows an issuing CA to indicate that, for the purposes of the user of a certification path containing this CA certificate, one of the issuer's certificate policies can be considered equivalent to a different certificate policy used in the subject CA's domain. This extension is defined as follows:

```
policyMappings EXTENSION ::= {
  SYNTAX          PolicyMappingsSyntax
  IDENTIFIED BY   id-ce-policyMappings }

PolicyMappingsSyntax ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
  issuerDomainPolicy    CertPolicyId,
  subjectDomainPolicy   CertPolicyId,
  ... }
```

The **issuerDomainPolicy** component indicates a certificate policy that is recognized in the issuing CA's domain and that can be considered equivalent to the certificate policy indicated in the **subjectDomainPolicy** component that is recognized in the subject CA's domain.

Policies shall not be mapped to or from the special value **anyPolicy**.

This extension may, at the option of the issuing CA, be either flagged as critical or as non-critical. It is recommended that it be flagged as critical, otherwise a relying party may not correctly interpret the stipulation of the issuing CA.

NOTE 1 – An example of policy mapping is as follows. The U.S. government domain may have a policy called Canadian Trade and the Canadian government may have a policy called U.S. Trade. While the two policies are distinctly identified and defined, there may be an agreement between the two governments to accept certification paths extending cross-border within the rules implied by these policies for relevant purposes.

NOTE 2 – Policy mapping implies significant administrative overheads and the involvement of suitably diligent and authorized personnel in related decision-making. In general, it is preferable to agree upon a more global use of common policies than it is to apply policy mapping. In the above example, it would be preferable for the U.S., Canada and Mexico to agree upon a common policy for North American Trade.

NOTE 3 – It is anticipated that policy mapping will be practical only in limited environments in which policy statements are very simple.

9.2.2.8 Authorization and validation extension

This extension may only be present in end-entity public-key certificates. It indicates that this public-key certification shall not be accepted if it cannot be checked against a particular authorization validation list (AVL).

```
authorizationValidation EXTENSION ::= {
  SYNTAX          AvlId
  IDENTIFIED BY   id-ce-authorizationValidation }

AvlId ::= SEQUENCE {
  issuer        Name,
  serialNumber  AvlSerialNumber OPTIONAL,
  ... }
```

The **issuer** component shall identify the issuer of the AVL against which this end-entity public-key certificate shall be checked.

The **serialNumber** component shall be present if the AVL includes the **serialNumber** component. Otherwise, it shall be absent.

This extension shall always be flagged as critical.

9.3 Subject and issuer information extensions

9.3.1 Requirements

The following requirements relate to public-key certificate **subject** component and public-key certificate and CRL **issuer** component:

a) Public-key certificates need to be usable by applications that employ a variety of name forms, including Internet electronic mail names, Internet domain names, Rec. ITU-T X.400 originator/recipient addresses, and EDI party names. It is therefore necessary to be able securely to associate multiple names of a variety of name forms with a public-key certificate subject or a public-key certificate or CRL issuer.

b) There may be a need for convoying identity and/or privilege information in a public-key certificate. Such information is provided as directory attributes.

9.3.2 Certificate and CRL extensions

The following extensions are defined:

a) Subject alternative name;

b) Issuer alternative name;

c) Subject directory attributes.

These extensions shall be used only as public-key certificate extensions, except for issuer alternative name, which may also be used as a CRL extension. As public-key certificate extensions, they may be present in CA certificates and end-entity public-key certificates.

9.3.2.1 Subject alternative name extension

This extension contains one or more alternative names, using any of a variety of name forms, for the entity that is bound by the issuing CA to the certified public key. This extension is defined as follows:

```
subjectAltName EXTENSION ::= {
  SYNTAX          GeneralNames
  IDENTIFIED BY   id-ce-subjectAltName }

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

GeneralName ::= CHOICE {
  otherName                  [0]  INSTANCE OF OTHER-NAME,
  rfc822Name                 [1]  IA5String,
  dNSName                    [2]  IA5String,
  x400Address                [3]  ORAddress,
  directoryName              [4]  Name,
  ediPartyName               [5]  EDIPartyName,
  uniformResourceIdentifier  [6]  IA5String,
  iPAddress                  [7]  OCTET STRING,
  registeredID               [8]  OBJECT IDENTIFIER,
  ... }
```

```
OTHER-NAME ::= TYPE-IDENTIFIER

EDIPartyName ::= SEQUENCE {
  nameAssigner   [0]  UnboundedDirectoryString OPTIONAL,
  partyName      [1]  UnboundedDirectoryString,
  ... }
```

The values in the alternatives of the **GeneralName** type are names of various forms as follows:

– the **otherName** alternative shall be a name of any form whose type is defined as an instance of the **OTHER-NAME** information object class;

– the **rfc822Name** alternative shall be an Internet electronic mail address defined in accordance with IETF RFC 822;

– the **dNSName** alternative shall be a fully qualified domain name (FQDN). The domain name shall be in the syntax as specified by section 2.3.1 of IETF RFC 5890 meaning that a domain name is a sequence of labels in the letters, digits, hyphen (LDH) format separated by dots.

  A label may be in one of two formats:

  a) All characters in the label are from the Basic Latin collection as defined by ISO/IEC 10646 (i.e., having code points in the ranges 002D, 0030-0039, 0041-005A and 0061-007A) and it does not start with "xn--". The maximum length is 63 octets.

  b) It is an A-label as defined in IETF RFC 5890, i.e., it starts with the "xn--" and is an U-label converted to valid ASCII characters as in item a) using the Punycode algorithm defined by IETF RFC 3492. The converted string shall be maximum 59 octets. To be valid, it shall be possible for an A-label to be convert a valid U-label. The U-label is as also defined in IETF RFC 5890.

  NOTE 1 – An A-label is normally not human readable.

– the **x400Address** alternative shall be an O/R address as defined by Rec. ITU-T X.411 | ISO/IEC 10021-4;

– the **directoryName** alternative shall be a distinguished name as defined by Rec. ITU-T X.501 | ISO/IEC 9594-2;

– the **ediPartyName** alternative shall be a name of a form agreed between communicating electronic data interchange (EDI) partners; the **nameAssigner** component identifies an authority that assigns unique values of names in the **partyName** component;

– the **uniformResourceIdentifier** alternative shall be a uniform resource identifier for the worldwide web defined in accordance with IETF RFC 1630;

– the **iPAddress** alternative shall be an Internet Protocol address defined in accordance with IETF RFC 791 for IPv4 (four octets) or in accordance with IETF RFC 2460 for IPv6 (16 octets);

– the **registeredID** alternative shall be an object identifier of any registered object assigned in accordance with Rec. ITU-T X.660 | ISO/IEC 9834-1.

For every name form used in an instance of the **GeneralName** data type, the issuing CA shall assure that it does not allocate the same name to different entities. A name of a particular type together with the identity of the issuing CA shall uniquely identify a particular entity.

This extension may, at the option of the issuing CA, be either flagged as critical or as non-critical. A relying party that supports this extension is not required to be able to process all name forms. If the extension is flagged as critical, at least one of the name forms that is present shall be recognized and processed, otherwise the public-key certificate shall be considered invalid. Apart from the preceding restriction, a relying party is permitted to ignore any name with an unrecognized or unsupported name form. It is recommended that, provided the subject component of the public-key certificate contains a distinguished name that unambiguously identifies the subject, this extension be flagged as non-critical.

NOTE 2 – Use of the **TYPE-IDENTIFIER** information object class is described in Rec. ITU-T X.681 | ISO/IEC 8824-2.

NOTE 3 – If this extension is present and is flagged as critical, the **subject** component of an end-entity public-key certificate may contain a null name (e.g., a sequence of zero relative distinguished names) in which case the subject is identified only by the name or names in this extension (see clause 7.2).

9.3.2.2 Issuer alternative name extension

This extension contains one or more alternative names, using any of a variety of name forms, for the CA or CRL issuer. This extension is defined as follows:

```
issuerAltName EXTENSION ::= {
  SYNTAX         GeneralNames
  IDENTIFIED BY  id-ce-issuerAltName }
```

This extension may, at the option of the CA or the CRL issuer, be flagged either as critical or as non-critical. A relying party that supports this extension is not required to be able to process all name forms. If the extension is flagged as critical, at least one of the name forms that are present shall be recognized and processed, otherwise the public-key certificate or CRL shall be considered invalid. Apart from the preceding restriction, a relying party is permitted to ignore any name with an unrecognized or unsupported name form. It is recommended that, provided the **issuer** component of the public-key certificate or CRL contains a distinguished name that unambiguously identifies the issuing authority, this extension be flagged as non-critical.

9.3.2.3 Subject directory attributes extension

This extension conveys directory attributes associated with the subject of the public-key certificate. This extension is defined as follows:

```
subjectDirectoryAttributes EXTENSION ::= {
  SYNTAX         AttributesSyntax
  IDENTIFIED BY  id-ce-subjectDirectoryAttributes }

AttributesSyntax ::= SEQUENCE SIZE (1..MAX) OF Attribute{{SupportedAttributes}}
```

This extension may, at the option of the issuing CA, be flagged either as critical or as non-critical. A relying party processing this extension is not required to understand all attribute types included in the extension. If the extension is flagged as critical, at least one of the attribute types contained in the extension shall be understood for the public-key certificate to be considered valid. If the extension is flagged as critical and none of the contained attribute types is understood, the public-key certificate shall be considered invalid.

A relying party may require that it understands all the attribute types to accept the public-key certificate.

This extension is intended to associate identity and/or privilege with the subject of the public-key certificate. When the subject accesses an entity by including identity information, the accessed entity may, based on local rules, assign privilege to the subject, e.g., assign privilege used for access control. Such identity information may be different from or a supplement to the identity information supplied in the **subject** component and/or in the **subjecatAltName** extension (if present).

If this extension is present in a public-key certificate and flagged as critical, some of the extensions defined in clause 17 may also be present as stated for the individual extensions.

9.4 Certification path constraint extensions

9.4.1 Requirements

For certification path processing:

a) End-entity public-key certificates need to be distinguishable from CA certificates to protect against entities from using their end-entity public-key certificates to establish themselves as CAs. It also needs to be possible for a CA to limit the length of a subsequent chain resulting from a certified subject CA, e.g., to no more than one more CA certificate or no more than two more CA certificates.

b) A CA needs to be able to specify constraints which allow a relying party to check that less-trusted CAs in a certification path (i.e., CAs further down the certification path from the CA with whose public key the relying party starts) are not violating their trust by issuing public-key certificates to subjects in an inappropriate name space. Adherence to these constraints needs to be automatically checkable by the relying party.

c) Certification path processing needs to be implementable in an automated, self-contained module. This is necessary to permit trusted hardware or software modules to be implemented which perform the certification path processing functions.

d) It should be possible to implement certification path processing without depending upon real-time interactions with a possible human user.

e) It should be possible to implement certification path processing without depending upon the use of trusted local databases of policy-description information. (Some trusted local information – an initial public key, at least – is needed for certification path processing but the amount of such information should be minimized.)

f) Certification paths need to operate in environments in which multiple certificate policies are recognized. A CA needs to be able to stipulate which CAs in other domains it trusts and for which purposes. Chaining through multiple policy domains needs to be supported.

g) Naming structures should not be constrained by the need to use names in public-key certificates, i.e., distinguished name structures considered natural for organizations or geographical areas shall not need adjustment in order to accommodate CA requirements.

h) Certificate extensions need to be backward-compatible with the unconstrained certification path approach system as specified in earlier editions of this Specification.

i) A CA needs to be able to inhibit the use of policy mapping and to require explicit certificate policy identifiers to be present in subsequent certificates in a certification path.

NOTE – In any relying party, the processing of a certification path requires an appropriate level of assurance. This Specification defines functions that may be used in implementations that are required to conform to specific assurance statements. For example, an assurance requirement could state that certification path processing shall be protected from subversion of the process (such as software-tampering or data modification). The level of assurance should be commensurate with business risk. For example:

– processing internal to an appropriate cryptographic module may be required for public keys used to validate high value funds transfer; whereas

– processing in software may be appropriate for home banking balance inquiries.

Consequently, certification path processing functions should be suitable for implementation in hardware cryptographic modules or cryptographic tokens as one option.

j) A CA needs to be able to prevent the special value **`anyPolicy`** from being considered a valid policy in subsequent public-key certificates in a certification path.

9.4.2 Public-key certificate extensions

The following extensions are defined:

a) **`basicConstraints`**;

b) **`nameConstraints`**;

c) **`policyConstraints`**; and

d) **`inhibitAnyPolicy`**.

These extensions shall be used only as public-key certificate extensions. Name constraints and policy constraints shall be used only in CA certificates; basic constraints may also be used in end-entity public-key certificates. Examples of the use of these extensions are given in Annex I.

9.4.2.1 Basic constraints extension

This extension indicates if the subject may act as a CA, with the certified public key being used to verify public-key certificate signatures. If so, a certification path length constraint may also be specified. This extension is defined as follows:

```
basicConstraints EXTENSION ::= {
  SYNTAX         BasicConstraintsSyntax
  IDENTIFIED BY  id-ce-basicConstraints }

BasicConstraintsSyntax ::= SEQUENCE {
  cA                   BOOLEAN DEFAULT FALSE,
  pathLenConstraint    INTEGER(0..MAX) OPTIONAL,
  ... }
```

The **`cA`** component with the value **`TRUE`** indicates that the certified public key may be used to verify public-key certificate signatures.

The **`pathLenConstraint`** component may be present if **`cA`** component is set to **`TRUE`**. Otherwise, it shall be absent. It gives the maximum number of CA certificates that may follow this CA certificate in a certification path. Value 0 indicates that the subject of this CA certificate may only issue end-entity public-key certificates, but shall not issue any CA certificate. If no **`pathLenConstraint`** extension appears in any CA certificate of a certification path, there is no limit to the allowed length of the certification path. The constraint takes effect beginning with the next CA certificate in the path. The constraint restricts the length of the segment of the certification path between the CA certificate containing this extension and the end-entity public-key certificate. It has no impact on the number of CA certificates in the certification path between the trust anchor and the CA certificate containing this extension. Therefore, the length of a complete certification path may exceed the maximum length of the segment constrained by this extension. The constraint controls the number of non-self-issued CA certificates between the CA certificate containing the constraint and the end-entity public-key certificate. Therefore, the total length of this segment of the path, excluding self-issued certificates, may exceed the value of the constraint by as many as two certificates. (This includes the public-key certificates at the two endpoints of the segment plus the CA certificates between the two endpoints that are constrained by the value of this extension.)

This extension shall be supported by a conformant relying party.

This extension shall be present in a CA certificate with the **cA** component set to **TRUE** and flagged as critical (which requires that a CA certificate shall be a version 3 public-key certificate).

This extension may, when included in an end-entity public-key certificate at the option of the issuing CA, be either flagged as critical or as non-critical.

When this extension is present, then:

– if the value of the **cA** component is not set to **TRUE** then the certified public key shall not be used to verify a public-key certificate signature;

– if the value of **cA** component is set to **TRUE** and **pathLenConstraint** is present then the relying party shall check that the certification path being processed is consistent with the value of **pathLenConstraint**.

If this extension is not present, then the public-key certificate is to be considered an end-entity public-key certificate and cannot be used to verify public-key certificate signatures.

NOTE – To constrain a public-key certificate subject to act only as an end entity, i.e., not as a CA, the issuer may include this extension containing only an empty **SEQUENCE** value.

9.4.2.2 Name constraints extension

This extension, which shall be used only in a CA certificate, indicates one or more name forms which have constraints placed upon their name spaces, and in which all subject names in the same name form in subsequent public-key certificates in a certification path shall be located. If this extension is absent, then no constraints are placed on any name form. If this extension is present but a name form is not included in the extension, then no constraints are imposed on that name form.

NOTE – Because there can be an unbounded set of **otherName** name forms, then in general it is not possible to constrain every possible name form of subject names with this extension.

This extension is defined as follows:

```
nameConstraints EXTENSION ::= {
  SYNTAX         NameConstraintsSyntax
  IDENTIFIED BY  id-ce-nameConstraints }

NameConstraintsSyntax ::= SEQUENCE {
  permittedSubtrees   [0]  GeneralSubtrees OPTIONAL,
  excludedSubtrees    [1]  GeneralSubtrees OPTIONAL,
  ... }
  (WITH COMPONENTS {..., permittedSubtrees  PRESENT } |
   WITH COMPONENTS {..., excludedSubtrees   PRESENT } )

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

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

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

At least one of **permittedSubtrees** and **excludedSubtrees components** shall be present.

NOTE – The subtree concept is briefly described in annex M.

The **permittedSubtrees** component, when present, shall specify one or more subtrees, for one or more name forms, within which subject names in acceptable public-key certificates shall be contained. When present, the **excludedSubtrees** component specifies one or more subtrees for one or more name forms within which subject names in acceptable public-key certificates shall not be contained. Subject names that are compared against specified subtrees include those present in both the **subject** component and the **subjectAltNames** extension of a public-key certificate. Each subtree is defined by the name of the root of the subtree, the **base** component, and, optionally, within that subtree, an area that is bounded by upper and/or lower levels.

The **minimum** component specifies the upper bound of the area within the subtree. All names whose final name component is above the level specified are not contained within the area. A value of **minimum** equal to zero (the default) corresponds to the base, i.e., the top node of the subtree. For example, if **minimum** is set to one, then the subtree excludes the base node but includes subordinate nodes.

The **maximum** component specifies the lower bound of the area within the subtree. All names whose last component is below the level specified are not contained within the area. A value of **maximum** of zero corresponds to the base, i.e., the top of the subtree. An absent **maximum** component indicates that no lower limit should be imposed on the area within the subtree. For example, if **maximum** is set to one, then the subtree excludes all nodes except the subtree base and its immediate subordinates.

The set of all **permittedSubtrees** and **excludedSubtrees** for a name form together constitute the constrained name space for the name form. All subject names, in public-key certificates issued by the subject CA and subsequent CAs in a certification path, which are of a constrained name form, shall be located in the constrained name space for the public-key certificate to be acceptable.

The **permittedSubtrees** component, when present, specifies the subtrees within which all the subject names that are of a constrained name form shall lie, for the public-key certificate to be acceptable. If **excludedSubtrees** is present, any public-key certificate issued by the subject CA or subsequent CAs in the certification path that has a subject name within these subtrees is unacceptable. If both **permittedSubtrees** and **excludedSubtrees** are present for a name form and the name spaces overlap, the exclusion statement takes precedence.

If none of the name forms of the subject name in the public-key certificate is constrained by this extension, the public-key certificate is acceptable from a name constraints point of view.

In some situations, more than one public-key certificate may need to be issued to satisfy the name constraints requirements. Annex I illustrates two of these situations. For example, if name constraints are defined for multiple name forms, but a public-key certificate needs to meet the name constraints for only one of the name forms (logical OR on constraints), then multiple public-key certificates should be issued, each constraining a single name form.

Of the name forms available through the **GeneralName** data type, only those name forms that have a well-defined hierarchical structure may be used in these components.

The **directoryName** name form satisfies this requirement; when using this name form a naming subtree corresponds to a directory information tree (DIT) subtree. A public-key certificate is considered subordinate to the **base** (and therefore a candidate to be within the subtree) if the **SEQUENCE** of **RDN**s, which forms the full distinguished name in **base**, is identical to the initial **SEQUENCE** of the same number of **RDN**s which forms the first part of the distinguished name of the subject (in the **subject** component or **directoryName** of **subjectAltNames** extension) of the public-key certificate. The distinguished name of the subject of the public-key certificate may have additional trailing **RDN**s in its sequence that do not appear in the distinguished name in **base**. The **distinguishedNameMatch** matching rule is used to compare the value of **base** with the initial sequence of **RDN**s in the distinguished name of the subject of the public-key certificate.

Conformant relying parties are not required to recognize all possible name forms. If the extension is flagged as critical and a relying party does not recognize a name form used in any **base** component, the public-key certificate shall be handled as if an unrecognized critical extension had been encountered. If the extension is flagged as non-critical and a relying party does not recognize a name form used in any **base** component, then that subtree may be ignored.

When testing public-key certificate subject names for consistency with a name constraint, names in non-critical subject alternative name extensions shall be processed, not ignored.

This extension may, at the option of the issuing CA, be flagged either as critical or as non-critical. It is recommended that it be flagged as critical; otherwise, a relying party may not check that subsequent public-key certificates in a certification path are located in the constrained name spaces intended by the issuing CA.

If this extension is present and is flagged as critical, then a relying party shall check that the certification path being processed is consistent with the value in this extension.

Annex I contains examples of the use of the **nameConstraints** extension.

9.4.2.3 Policy constraints extension

This extension specifies constraints which may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path. This extension is defined as follows:

```
policyConstraints EXTENSION ::= {
  SYNTAX          PolicyConstraintsSyntax
  IDENTIFIED BY  id-ce-policyConstraints }

PolicyConstraintsSyntax ::= SEQUENCE {
  requireExplicitPolicy   [0]  SkipCerts OPTIONAL,
  inhibitPolicyMapping    [1]  SkipCerts OPTIONAL,
  ... }
  (WITH COMPONENTS {..., requireExplicitPolicy PRESENT } |
   WITH COMPONENTS {..., inhibitPolicyMapping  PRESENT } )

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

At least one of the **`requireExplicitPolicy`** and the **`inhibitPolicyMapping`** components shall be present.

If the **`requireExplicitPolicy`** component is present, and the certification path includes a public-key certificate issued by a nominated CA, it is necessary for all public-key certificates in the path to contain, in the certificate policies extension, an acceptable policy identifier. An acceptable policy identifier is the identifier of a certificate policy required by the relying party, the identifier of a policy which has been declared equivalent to one of these policies through policy mapping, or **`anyPolicy`**. The nominated CA is either the issuer CA of the public-key certificate containing this extension (if the value of **`requireExplicitPolicy`** is 0) or a CA which is the issuer of a subsequent public-key certificate in the certification path (as indicated by a non-zero value).

If the **`inhibitPolicyMapping`** component is present, it indicates that, in all public-key certificates starting from a nominated CA in the certification path until the end of the certification path, policy mapping is not permitted. The nominated CA is either the subject CA of the public-key certificate containing this extension (if the value of **`inhibitPolicyMapping`** is 0) or a CA which is the subject of a subsequent public-key certificate in the certification path (as indicated by a non-zero value).

A value of type **`SkipCerts`** indicates the number of public-key certificates in the certification path to skip before a constraint becomes effective.

This extension may, at the option of the issuing CA, be flagged either as critical or as non-critical. It is recommended that it be flagged as critical; otherwise, a relying party may not correctly interpret the stipulation of the issuing CA.

9.4.2.4 Inhibit any policy extension

This extension specifies a constraint that indicates **`anyPolicy`** is not considered an explicit match for other certificate policies for all non-self-issued public-key certificates in the certification path starting with a nominated CA. The nominated CA is either the subject CA of the CA certificate containing this extension (if the value of **`inhibitAnyPolicy`** is 0) or a CA which is the subject of a subsequent CA certificate in the certification path (as indicated by a non-zero value).

```
inhibitAnyPolicy EXTENSION ::= {
  SYNTAX          SkipCerts
  IDENTIFIED BY   id-ce-inhibitAnyPolicy }
```

This extension may, at the option of the issuing CA, be flagged as either critical or as non-critical. It is recommended that it be flagged as critical, otherwise a relying party may not correctly interpret the stipulation of the issuing CA.

9.5 Basic CRL extensions

9.5.1 Requirements

The following requirements relate to CRLs:

a) Relying parties need to be able to track all CRLs issued from a CRL issuer or CRL distribution point (see clause 9.6) and be able to detect a missing CRL in the sequence. CRL sequence numbers are therefore required.

b) Some relying parties may wish to respond differently to a revocation, depending upon the reason for the revocation. There is therefore a requirement for a CRL entry to indicate the reason for the revocation.

c) There is a requirement for an authority to be able to temporarily suspend validity of a public-key certificate and subsequently either revoke or reinstate it. Possible reasons for such an action include:

   – desire to reduce liability for erroneous revocation when a revocation request is unauthenticated and there is inadequate information to determine whether it is valid;

   – other business needs, such as temporarily disabling the public-key certificate of an entity pending an audit or investigation.

d) A CRL contains, for each public-key revoked certificate, the date when the authority posted the revocation. Further information may be known as to when an actual or suspected key compromise occurred, and this information may be valuable to a relying party. The revocation date is insufficient to solve some disputes because, assuming the worst, all signatures issued during the validity period of the public-key certificate have to be considered invalid. However, it may be important for a user that a signed document be recognized as valid even though the key used to sign the message was compromised after the signature was produced. To assist in solving this problem, a CRL entry can include a second date which indicates when it was known or suspected that the private key was compromised.

e) Relying parties need to be able to determine, from the CRL itself, additional information including the scope of public-key certificates covered by this list, the ordering of revocation notices, and which stream of CRLs the CRL number is unique within.

f) Issuers need the ability dynamically to change the partitioning of CRLs and to refer relying parties to the new location for relevant CRLs if the partitioning changes.

g) Delta CRLs may also be available that update a given base CRL. Relying parties need to be able to determine, from a given CRL, whether delta CRLs are available, where they are located and when the next delta CRL will be issued.

h) In addition to CRLs publishing a notification that public-key certificates have been revoked, there is a requirement to publish a notification that public-key certificates will be revoked as of a specified date and time in the future.

i) There is a requirement to provide more efficient ways to indicate in a CRL that a set of public-key certificates has been revoked.

9.5.2 CRL extensions

The following extensions are defined:

a) CRL number;

b) status referral;

c) CRL stream identifier;

d) ordered list;

e) delta information;

f) to be revoked;

g) revoked group of certificates; and

h) expired certificates on CRL.

9.5.2.1 CRL number extension

This CRL extension conveys a monotonically increasing sequence number for each CRL issued by a given CRL issuer through a given CRL directory attribute (see clauses 13.2.4 to 13.2.7) or CRL distribution point. It allows a relying party to detect whether CRLs issued prior to the one being processed were also seen and processed. This extension is defined as follows:

```
cRLNumber EXTENSION ::= {
  SYNTAX          CRLNumber
  IDENTIFIED BY  id-ce-cRLNumber }

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

NOTE - The extension defined here is relevant for both CRLs and ACRLs. Only CRL aspects are considered here. For ACRL aspects, see 17.7.2.1.

This extension shall always be flagged as non-critical.

9.5.2.2 Status referral extension

This CRL extension is for use within the CRL structure as a means to convey information about revocation notices to relying parties. As such, it would be present in a CRL structure that itself contains no certificate revocation notices. A CRL structure containing this extension shall not be used by relying parties or relying parties as a source of revocation notices, but rather as a tool to ensure that the appropriate revocation information is used. Any CRL containing this extension shall not be used as the source for a relying party to check revocation status of any public-key certificate. Rather, a CRL containing this extension may be used by a relying party as an additional tool to locate the appropriate CRLs for checking revocation status.

This extension serves two primary functions:

– This extension provides a mechanism to publish a trusted "list of CRLs" including all the relevant information to aid relying parties in determining whether they have sufficient revocation information for their needs. For example, an authority may issue a new, authenticated CRL list periodically, typically with a relatively high reissue frequency (in comparison with other CRL reissue frequencies). The list might include a last-update time/date for every referenced CRL. A relying party, on obtaining this list, can quickly determine if cached copies of CRLs are still up-to-date. This may eliminate the unnecessary retrieval of CRLs. Furthermore, by using this mechanism, relying parties become aware of CRLs issued by the authority between its usual update cycles, thereby improving the timeliness of the CRL system.

– This extension also provides a mechanism to redirect a relying party from a preliminary location (e.g., one pointed to in a CRL distribution point extension, or the directory entry of the issuing authority) to a different location for revocation information. This feature enables authorities to modify the CRL partitioning scheme they use without impacting existing public-key certificates or relying parties. To achieve this, the authority would include each new location and the scope of the CRL that would be found at that location. The relying party would compare the public-key certificate of interest with the scope statements and follow the pointer to the appropriate new location for revocation information relevant to that public-key certificate it is validating.

The extension is itself extensible and in future other non-CRL based revocation schemes may also be referred to, using this extension.

```
statusReferrals EXTENSION ::= {
  SYNTAX         StatusReferrals
  IDENTIFIED BY  id-ce-statusReferrals }

StatusReferrals ::= SEQUENCE SIZE (1..MAX) OF StatusReferral

StatusReferral ::= CHOICE {
  cRLReferral    [0]  CRLReferral,
  otherReferral  [1]  INSTANCE OF OTHER-REFERRAL,
  ... }

CRLReferral ::= SEQUENCE {
  issuer          [0]  GeneralName OPTIONAL,
  location        [1]  GeneralName OPTIONAL,
  deltaRefInfo    [2]  DeltaRefInfo OPTIONAL,
  cRLScope             CRLScopeSyntax,
  lastUpdate      [3]  GeneralizedTime OPTIONAL,
  lastChangedCRL  [4]  GeneralizedTime OPTIONAL,
  ...
}

DeltaRefInfo ::= SEQUENCE {
  deltaLocation  GeneralName,
  lastDelta      GeneralizedTime OPTIONAL,
  ... }

OTHER-REFERRAL ::= TYPE-IDENTIFIER
```

NOTE - The extension defined here is relevant for both CRLs and ACRLs. Only CRL aspects are considered here. For ACRL aspects, see 17.7.2.2.

The **issuer** component identifies the entity that signed the CRL; this defaults to the issuer name of the encompassing CRL.

The **location** component provides the location to which the referral is to be directed, and defaults to the same value as the **issuer** name.

The **deltaRefInfo** component provides an optional alternative location from which a dCRL may be obtained and an optional date of the previous delta.

The **cRLScope** component provides the scope of the CRL that will be found at the referenced location.

The **lastUpdate** component is the value of the **thisUpdate** component in the most recently issued referenced CRL.

The **lastChangedCRL** component is the value of the **thisUpdate** component in the most recently issued CRL that has changed content.

The **OTHER-REFERRAL** provides extensibility to enable other non-CRL based revocation schemes to be accommodated in future.

This extension shall always be flagged as critical to ensure that the CRL containing this extension is not inadvertently relied on by relying parties as the source of revocation status information about public-key certificates.

If this extension is present and is recognized by a relying party, that system shall not use the CRL as a source of revocation status information. The system should use either the information contained in this extension, or other means outside the scope of this Specification, to locate appropriate revocation status information.

If this extension is present but is not recognized by a relying party, that system shall not use the CRL as a source of revocation status information. The system should use other means, outside the scope of this Specification, to locate appropriate revocation information.

9.5.2.3 CRL stream identifier extension

The CRL stream identifier extension is used to identify the context within which the CRL number is unique.

```
cRLStreamIdentifier EXTENSION ::= {
  SYNTAX         CRLStreamIdentifier
  IDENTIFIED BY  id-ce-cRLStreamIdentifier }

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

NOTE - The extension defined here is relevant for both CRLs and ACRLs. Only CRL aspects are considered here. For ACRL aspects, see 17.7.2.3.

This extension shall always be flagged as non-critical.

Each value of this extension, per authority, shall be unique. The CRL stream identifier combined with a CRL number serve as a unique identifier for each CRL issued by any given authority, regardless of the type of CRL.

9.5.2.4 Ordered list extension

The ordered list extension indicates that the sequence of revoked public-key certificates in the **revokedCertificates** component of a CRL is in ascending order by either public-key certificate serial number or revocation date. This extension is defined as follows:

```
orderedList EXTENSION ::= {
  SYNTAX         OrderedListSyntax
  IDENTIFIED BY  id-ce-orderedList }

OrderedListSyntax ::= ENUMERATED {
  ascSerialNum (0),
  ascRevDate   (1),
  ...}
```

NOTE - The extension defined here is relevant for both CRLs and ACRLs. Only CRL aspects are considered here. For ACRL aspects, see 17.7.2.4.

This extension shall always be flagged as non-critical.

– **ascSerialNum** indicates that the sequence of revoked public-key certificates in a CRL is in ascending order of public-key certificate serial number, based on the value of the **serialNumber** component of each entry in the list.

– **ascRevDate** indicates that the sequence of revoked public-key certificates in a CRL is in ascending order of revocation date, based on the value of the **revocationDate** component of each entry in the list.

If **orderedList** is not present, no information is provided as to the ordering, if any, of the list of revoked public-key certificates in the CRL.

9.5.2.5 Delta Information extension

This CRL extension is for use in CRLs that are not dCRLs and is used to indicate to relying parties that dCRLs are also available for the CRL containing this extension. The extension provides the location at which the related dCRLs can be found and optionally the time at which the next dCRL is to be issued.

```
deltaInfo EXTENSION ::= {
  SYNTAX         DeltaInformation
  IDENTIFIED BY  id-ce-deltaInfo }

DeltaInformation ::= SEQUENCE {
  deltaLocation  GeneralName,
  nextDelta      GeneralizedTime OPTIONAL,
  ... }
```

NOTE - The extension defined here is relevant for both CRLs and ACRLs. Only CRL aspects are considered here. For ACRL aspects, see 17.7.2.5.

This extension shall always be flagged as non-critical.

9.5.2.6 To be revoked extension

This CRL extension allows for the notification that public-key certificates will be revoked as of a specified date and time in the future. The **toBeRevoked** extension is used to specify the reason for the public-key certificate revocation, the date and time at which the public-key certificate will be revoked, and the group of public-key certificates to be revoked. Each list can contain a single public-key certificate serial number, a range of public-key certificate serial numbers or a named **subtree**.

```
toBeRevoked EXTENSION ::= {
  SYNTAX          ToBeRevokedSyntax
  IDENTIFIED BY   id-ce-toBeRevoked }

ToBeRevokedSyntax ::= SEQUENCE SIZE (1..MAX) OF ToBeRevokedGroup

ToBeRevokedGroup ::= SEQUENCE {
  certificateIssuer   [0]  GeneralName OPTIONAL,
  reasonInfo          [1]  ReasonInfo OPTIONAL,
  revocationTime           GeneralizedTime,
  certificateGroup        ,
  ... }

ReasonInfo ::= SEQUENCE {
  reasonCode            CRLReason,
  holdInstructionCode   HoldInstruction OPTIONAL,
  ... }

CertificateGroup ::= CHOICE {
  serialNumbers       [0]  CertificateSerialNumbers,
  serialNumberRange   [1]  CertificateGroupNumberRange,
  nameSubtree         [2]  GeneralName,
  ... }

CertificateGroupNumberRange ::= SEQUENCE {
  startingNumber   [0]  INTEGER,
  endingNumber     [1]  INTEGER,
  ... }

CertificateSerialNumbers ::= SEQUENCE SIZE (1..MAX) OF CertificateSerialNumber
```

NOTE - The extension defined here is relevant for both CRLs and ACRLs. Only CRL aspects are considered here. For ACRL aspects, see 17.7.2.6

The **certificateIssuer** component, when present, identifies the CA that issued all the public-key certificates listed in this **ToBeRevokedGroup**. If **certificateIssuer** is omitted, it defaults to the CRL issuer name.

The **reasonInfo** component, when present, identifies the reason for the public-key certificate revocations. When present, this extension indicates that all public-key certificates identified in **ToBeRevokedGroup** will be revoked for the reason indicated in this component. If **reasonCode** component contains the value **certificateHold**, the **holdInstructionCode** component may also be present. When present, the **holdInstructionCode** component indicates the action to be taken on encountering any of the public-key certificates identified in **RevokedGroup**. This action should only be taken, after the revocation time indicated in the **revocationTime** component has passed.

The **revocationTime** component indicates the date and time at which this group of public-key certificates will be revoked and should therefore be considered invalid. This date shall be later than the **thisUpdate** time of the CRL containing this extension. If **revocationTime** is before the **nextUpdate** time of the CRL containing this extension, the public-key certificates shall be considered revoked between the **revocationTime** and the **nextUpdate** time by a relying party using a CRL containing this extension. Otherwise, this is a notice that at specified time in the future these public-key certificates will be revoked. Once the revocation time has passed, either the CA has revoked the public-key certificate or not. If it has revoked the public-key certificate, future CRLs shall include this on the list of revoked public-key certificates, at least until the public-key certificate expires. If the CA has not revoked the public-key certificate, but still intends to revoke it in the future, it may include the public-key certificate in this extension on subsequent CRLs with a revised **revocationTime**. If the CA no longer intends to revoke the public-key certificate, it may be excluded from all subsequent CRLs and the public-key certificate shall not be considered revoked.

The **certificateGroup** component lists the set of public-key certificates to be revoked. This component identifies the public-key certificates issued by the CA identified in **certificateIssuer** to be revoked at the date/time identified in **revocationTime**. This set of public-key certificates is not further refined by any outside controls (e.g., **issuingDistributionPoint**).

The **serialNumbers** component, when present, shall hold the serial number(s) of the public-key certificate(s) issued by the identified issuing CA that will be revoked at the specified time.

The **serialNumberRange** component, when present, all public-key certificates in the range beginning with the starting serial number and ending with the ending serial number and issued by the identified issuing CA will be revoked at the specified time.

If the **nameSubtree** component is present, all public-key certificates with a subject name that is subordinate to the specified name and issued by the identified issuing CA will be revoked at the specified time. If the **nameSubtree** contains a distinguished name then all distinguished names associated with the subject of a public-key certificate (i.e., **subject** component and **subjectAltNames** extension) need to be considered. For other name forms, the **subjectAltNames** extension of public-key certificates need to be considered. If at least one of the names associated with the subject contained in the public-key certificate is within the subtree specified in **nameSubtree**, that public-key certificate will be revoked at the specified time. As with the **nameConstraints** extension, not all name forms are appropriate for **subtree** specification. Only those that have recognized subordination rules should be used in this extension.

This extension may, at the option of the CRL issuer, be flagged as critical or as non-critical. As the information provided in this extension applies to revocations, which will occur in the future, it is recommended that it be flagged as non-critical, reducing the risk of problems with interoperability and backward compatibility.

9.5.2.7 Revoked group of certificates extension

A set of public-key certificates that have been revoked can be published using the following CRL extension. Each list of public-key certificates to be revoked is associated with a specific public-key certificate issuing CA and revocation time. Each list can contain a range of public-key certificate serial numbers or a named subtree.

```
revokedGroups EXTENSION ::= {
  SYNTAX          RevokedGroupsSyntax
  IDENTIFIED BY  id-ce-revokedGroups }

RevokedGroupsSyntax ::= SEQUENCE SIZE (1..MAX) OF RevokedGroup

RevokedGroup ::= SEQUENCE {
  certificateIssuer         [0]  GeneralName OPTIONAL,
  reasonInfo                [1]  ReasonInfo OPTIONAL,
  invalidityDate            [2]  GeneralizedTime OPTIONAL,
  revokedcertificateGroup   [3]  RevokedCertificateGroup,
  ... }

RevokedCertificateGroup ::= CHOICE {
  serialNumberRange   NumberRange,
  nameSubtree         GeneralName }
```

NOTE - The extension defined here is relevant for both CRLs and ACRLs. Only CRL aspects are considered here. For ACRL aspects, see 17.7.2.7.

The **certificateIssuer** component, when present, shall identify the CA that issued all the public-key certificates listed in this **RevokedGroup**. If **certificateIssuer** is omitted, it defaults to the CRL issuer name.

The **reasonInfo** component, when present, shall identify the reason for the public-key certificate revocations. When present, this component indicates that all public-key certificates identified in **RevokedGroup** component were revoked for the reason indicated in this component. If **reasonCode** contains the value **certificateHold**, the **holdInstructionCode** may also be present. When present, **holdInstructionCode** shall indicate the action to be taken on encountering any of the public-key certificates identified in **RevokedGroup**.

The **invalidityDate** component, when present, indicates the time from which all public-key certificates identified in **RevokedGroup** should be considered invalid. This date shall be earlier than the date contained in **thisUpdate** component of the CRL. If omitted, all public-key certificates identified in **RevokedGroup** should be considered invalid at least from the time indicated in the **thisUpdate** component of the CRL. If the status of the public-key certificate prior to the **thisUpdate** time is critical to a relying party (e.g., to determine whether a digital signature that was created prior to this CRL issuance occurred while the public-key certificate was still valid or after it had been revoked), additional revocation status checking techniques will be required to determine the actual date/time from which a given public-key certificate should be considered invalid.

The **revokedCertificateGroup** component shall list the set of public-key certificates that have been revoked. This component identifies the public-key certificates issued by the CA identified in **certificateIssuer** revoked under the specified conditions. This set of public-key certificates is not further refined by any outside controls (e.g., **issuingDistributionPoint**).

The **serialNumberRange**, when present, shall specify a specified range within which all public-key certificates containing serial numbers and which are issued by the identified issuing CA are applicable.

If **nameSubtree** is present, all public-key certificates with a subject name that is subordinate to the specified name and

issued by the identified issuing CA will be revoked at the specified time. If the **nameSubtree** contains a distinguished name then all distinguished names associated with the subject of a public-key certificate (i.e., **subject** component and **subjectAltNames** extension) need to be considered. For other name forms, the **subjectAltNames** extension of public-key certificates need to be considered. If at least one of the names associated with the subject, contained in the public-key certificate, is within the subtree specified in **nameSubtree**, that public-key certificate has been revoked. As with the **nameConstraints** extension, not all name forms are appropriate for subtree specification. Only those that have recognized subordination rules should be used in this extension.

This extension shall always be flagged as critical. Otherwise, a relying party may incorrectly assume that public-key certificates, identified as revoked within this extension, are not revoked. When this extension is present it may be the only indication of revoked public-key certificates in a CRL (i.e., the **revokedCertificates** may be empty) or it may list revoked public-key certificates that are in addition to those indicated in the **revokedCertificates** component of this CRL. A revoked public-key certificate shall not be listed both in the **revokedCertificates** component and in this extension.

9.5.2.8 Expired certificates on CRL extension

This CRL extension indicates that the CRL includes revocation notices for expired public-key certificates.

```
expiredCertsOnCRL EXTENSION ::= {
  SYNTAX          ExpiredCertsOnCRL
  IDENTIFIED BY   id-ce-expiredCertsOnCRL }

ExpiredCertsOnCRL ::= GeneralizedTime
```

NOTE - The extension defined here is relevant for both CRLs and ACRLs. Only CRL aspects are considered here. For ACRL aspects, see 17.7.2.8.

This extension shall always be flagged as non-critical.

The scope of a CRL containing this extension is extended to include the revocation status of revoked public-key certificates that expired after the date specified in **ExpiredCertsOnCRL** or at that date. The revocation status of a public-key certificate shall not be updated once the public-key certificate has expired.

9.5.3 CRL entry extension

The following extensions are defined:

a) reason code;

b) hold instruction code; and

c) invalidity date.

9.5.3.1 Reason code extension

This CRL entry extension identifies a reason for the public-key certificate revocation. The reason code may be used by applications to decide, based on local policy, how to react to posted revocations. This extension is defined as follows:

```
reasonCode EXTENSION ::= {
  SYNTAX          CRLReason
  IDENTIFIED BY   id-ce-reasonCode }

CRLReason ::= ENUMERATED {
  unspecified            (0),
  keyCompromise          (1),
  cACompromise           (2),
  affiliationChanged     (3),
  superseded             (4),
  cessationOfOperation (5),
  certificateHold        (6),
  removeFromCRL          (8),
  privilegeWithdrawn     (9),
  aACompromise           (10),
  ...,
  weakAlgorithmOrKey     (11) }
```

NOTE - The extension defined here is relevant for both CRLs and ACRLs entries. Only CRL entry aspects are considered here. For ACRL entry aspects, see 17.7.3.1.

The following reason code values indicate why a public-key certificate was revoked:

- **`unspecified`** can be used to revoke public-key certificates for reasons other than the specific codes.
- **`keyCompromise`** is used in revoking an end-entity public-key certificate; it indicates that it is known or suspected that the subject's private key, or other aspects of the subject validated in the public-key certificate, have been compromised.
- **`cACompromise`** is used in revoking a CA certificate; it indicates that it is known or suspected that the subject's private key, or other aspects of the subject validated in the CA certificate, have been compromised.
- **`affiliationChanged`** indicates that the subject's name or other information in the public-key certificate has been modified but there is no cause to suspect that the private key has been compromised.
- **`superseded`** indicates that the public-key certificate has been superseded but there is no cause to suspect that the private key has been compromised.
- **`cessationOfOperation`** indicates that the public-key certificate is no longer needed for the purpose for which it was issued but there is no cause to suspect that the private key has been compromised.
- **`privilegeWithdrawn`** indicates that a public-key certificate was revoked because a privilege contained within that public-key certificate has been withdrawn.
- **`aACompromise`** is only relevant for ACRL entries (see 17.7.3.1).
- **`weekAlgorithm`** indicates that the public-key certificate was revoked due to a weak cryptographic algorithm and/or key (e.g., due to short key length or unsafe key generation).

A public-key certificate may be placed on hold by issuing a CRL entry with a reason code of **`certificateHold`**. The public-key certificate hold notice may include an optional hold instruction code to convey additional information to relying parties (see clause 9.5.3.2). Once a hold has been issued, it may be handled in one of three ways:

a) it may remain on the CRL with no further action, causing relying parties to consider the public-key certificate invalid during the hold period;

b) it may be replaced by a (final) revocation for the same public-key certificate, in which case the reason shall be one of the standard reasons for revocation, the revocation date shall be the date the public-key certificate was placed on hold, and the optional hold instruction code extension shall not appear;

c) it may be explicitly released and the entry removed from the CRL.

The **`removeFromCRL`** reason code is for use with delta CRLs (see clause 9.6) only and indicates that an existing CRL entry should now be removed owing to public-key certificate expiration or hold release. An entry with this reason code shall be used in delta CRLs for which the corresponding base CRL or any subsequent (delta or complete for scope) CRL contains an entry for the same public-key certificate with reason code **`certificateHold`**.

This extension shall always be flagged as non-critical.

9.5.3.2 Hold instruction code extension

This CRL entry extension provides for the inclusion of a registered instruction identifier to indicate the action to be taken on encountering a held public-key certificate. It is applicable only in a CRL entry having a **`certificateHold`** reason code. This extension is defined as follows:

```
holdInstructionCode EXTENSION ::= {
  SYNTAX          HoldInstruction
  IDENTIFIED BY   id-ce-holdInstructionCode }

HoldInstruction ::= OBJECT IDENTIFIER
```

This extension shall be always flagged as non-critical. No hold instruction codes are defined in this Specification.

NOTE 1 – The extension defined here is relevant for both CRLs and ACRLs entries. Only CRL entry aspects are considered here. For ACRL entry aspects (see 17.7.3.2).

NOTE 2 – Examples of hold instructions might be "please communicate with the CA" or "repossess the user's token".

9.5.3.3 Invalidity date extension

This CRL entry extension indicates the date at which it is known or suspected that the private key was compromised or that the public-key certificate should otherwise be considered invalid. This date may be earlier than the revocation date in the CRL entry, which is the date at which the authority processed the revocation. This extension is defined as follows:

```
invalidityDate EXTENSION ::= {
  SYNTAX          GeneralizedTime
  IDENTIFIED BY   id-ce-invalidityDate }
```

This extension shall always be flagged as non-critical.

NOTE 1 – The date in this extension is not, by itself, sufficient for non-repudiation purposes. For example, this date may be a date advised by the private key holder, and it is possible for such a person fraudulently to claim that a key was compromised sometime in the past, in order to repudiate a validly generated signature.

NOTE 2 – When a revocation is first posted by an authority in a CRL, the invalidity date may precede the date of issue of earlier CRLs. The revocation date should not precede the date of issue of earlier CRLs.

9.6 CRL distribution points and delta CRL extensions

9.6.1 Requirements

As it is possible for revocation lists to become large and unwieldy, the ability to represent partial CRLs is required. Different solutions are needed for two different types of implementations that process CRLs.

The first type of implementation is in individual workstations, possibly in an attached cryptographic token. These implementations are likely to have limited, if any, trusted storage capacity. Therefore the entire CRL would need to be examined to determine if it is valid, and then to see if the public-key certificate is valid. This processing could be lengthy if the CRL is long. Partitioning of CRLs is required to eliminate this problem for these implementations.

The second type of implementation is on high performance servers where a large volume of messages is processed, e.g., a transaction processing server. In this environment, CRLs are typically processed as a background task where, after the CRL is validated, the contents of the CRL are stored locally in a representation which expedites their examination, e.g., one bit for each public-key certificate indicating if it has been revoked. This representation is held in trusted storage. This type of server will typically require up-to-date CRLs for a large number of CAs. Since it already has a list of previously revoked public-key certificates, it only needs to retrieve a list of newly revoked public-key certificates. This list, called a dCRL, will be smaller and require fewer resources to retrieve and process than a complete CRL.

The following requirements therefore relate to CRL distribution points and dCRLs:

a) In order to control CRL sizes, it needs to be possible to assign subsets of the set of all public-key certificates issued by one CA to different CRLs. This can be achieved by associating every public-key certificate with a CRL distribution point which is either:
   – a directory entry whose CRL attribute will contain a revocation entry for that public-key certificate, if it has been revoked; or
   – a location such as an electronic mail address or Internet uniform resource identifier from which the applicable CRL can be obtained.

b) For performance reasons, it is desirable to reduce the number of CRLs that need to be checked when validating multiple public-key certificates, e.g., a certification path. This can be achieved by having one CRL issuer sign and issue CRLs containing revocations from multiple CAs.

c) There is a requirement for separate CRLs covering revoked CA certificates and revoked end-entity public-key certificates. This facilitates the processing of certification paths as the CRL for revoked CA certificates can be expected to be very short (usually empty). The **certificateRevocationList**, the **eepkCertificateRevocationList** and the **authorityRevocationList** attribute types have been specified for this purpose. However, for this separation to be secure, it is necessary to have an indicator in a CRL identifying which list it is. Otherwise, illegitimate substitution of one list for the other cannot be detected.

d) Provision is needed for a different CRL to exist for potential compromise situations (when there is a significant risk of private key misuse) than that including all routine binding terminations (when there is no significant risk of private key misuse).

e) Provision is also needed for partial CRLs (known as dCRLs) which only contain entries for public-key certificates that have been revoked since the issuance of a base CRL.

f) For dCRLs, provision is needed to indicate the date/time after which this list contains updates.

g) There is a requirement to indicate within a public-key certificate, where to find the freshest CRL (e.g., the most recent dCRL).

9.6.2 CRL distribution point and delta CRL extensions

The following extensions are defined:

a) CRL distribution points;

b) issuing distribution point;

c) certificate issuer;

d) delta CRL indicator;

e) base update; and

f) freshest CRL.

CRL distribution points and freshest CRL shall be used only as a public-key certificate extension. Issuing distribution point, delta CRL indicator and base update shall be used only as CRL extensions. Certificate issuer shall be used only as a CRL entry extension.

9.6.2.1 CRL distribution points extension

The CRL distribution points extension shall be used only as a public-key certificate extension and may be used in CA certificates and in end-entity public-key certificates. This extension identifies the CRL distribution point or points to which a relying party should refer to ascertain if a public-key certificate has been revoked. A relying party can obtain a CRL from an applicable distribution point or it may be able to obtain a current complete CRL from the CA directory entry.

This extension is defined as follows:

```
cRLDistributionPoints EXTENSION ::= {
  SYNTAX         CRLDistPointsSyntax
  IDENTIFIED BY  id-ce-cRLDistributionPoints }

CRLDistPointsSyntax ::= SEQUENCE SIZE (1..MAX) OF DistributionPoint

DistributionPoint ::= SEQUENCE {
  distributionPoint   [0]  DistributionPointName OPTIONAL,
  reasons             [1]  ReasonFlags OPTIONAL,
  cRLIssuer           [2]  GeneralNames OPTIONAL,
  ... }

DistributionPointName ::= CHOICE {
  fullName                   [0]  GeneralNames,
  nameRelativeToCRLIssuer    [1]  RelativeDistinguishedName,
  ... }

ReasonFlags ::= BIT STRING {
  unused                  (0),
  keyCompromise           (1),
  cACompromise            (2),
  affiliationChanged      (3),
  superseded              (4),
  cessationOfOperation    (5),
  certificateHold         (6),
  privilegeWithdrawn      (7),
  aACompromise            (8),
  weakAlgorithmOrKey      (9) }
```

NOTE 1 - The extension defined here is relevant for both CRLs and ACRLs). Only CRL aspects are considered here. For ACRL aspects, see 17.2.2.1.

The **distributionPoint** component, when present, shall identify the location from which the relevant CRL can be obtained. If this component is absent, the distribution point name defaults to the CRL issuer name. This component has the following two alternatives:

– When the **fullName** alternative is used or when the default applies, the distribution point name may have multiple name forms. The same name, in at least one of its name forms, shall be present in the **distributionPoint** component of the **issuingDistributionPoint** extension of the CRL. A relying party is not required to be able to process all name forms. It may use a distribution point provided at least one name form can be processed. If no name forms for a distribution point can be processed, a relying party can still use the public-key certificate provided requisite revocation information can be obtained from another source, e.g., another distribution point or the authority's directory entry.

– The **nameRelativeToCRLIssuer** alternative can be used only if the CRL distribution point is assigned a distinguished name that is directly subordinate to the distinguished name of the CRL issuer. In this case, the **nameRelativeToCRLIssuer** component conveys the relative distinguished name with respect to the CRL issuer distinguished name.

The **reasons** component, when present, shall indicate the revocation reasons covered by this CRL. If the **reasons** component is absent, the corresponding CRL distribution point distributes a CRL which will contain an entry for this public-key certificate if this public-key certificate has been revoked, regardless of revocation reason. Otherwise, the **reasons** component indicates which revocation reasons are covered by the corresponding CRL distribution point.

NOTE 2 – While this components allows for shorter CRLs, it has the side-effect that a relying party has to search multiple CRLs to ensure that a particular public-key certificate has not been revoked. This possibility for segmenting CRLs should be used with caution.

The **cRLIssuer** component, when present, shall identify the authority that issued and signed the CRL. If this component is absent, the CRL issuer name defaults to the public-key certificate issuer name.

This extension may, at the option of the CA, be either flagged as critical or as non-critical. In the interests of interoperability, it is recommended that it be flagged as non-critical.

If this extension is flagged as critical, then a relying party shall not use the public-key certificate without first retrieving and checking a CRL from one of the nominated distribution points covering the reason codes of interest. Where the distribution points are used to distribute CRL information for all revocation reason codes and for all public-key certificates issued by the CA include this extension as a critical extension, the CA is not required to also publish a full CRL at the CA entry.

If this extension is flagged as non-critical and a relying party does not recognize the extension type, then that relying party should only consider the public-key certificate valid if:

– it can acquire and check a complete CRL from the authority (that the latter CRL is complete is indicated by the absence of an issuing distribution point extension in the CRL);

– revocation checking is not required under local policy; or

– revocation checking is accomplished by other means.

NOTE 3 – It is possible to have CRLs issued by more than one CRL issuer for the one public-key certificate. Coordination of these CRL issuers and the issuing authority is an aspect of authority policy.

NOTE 4 – The meaning of each reason code is as defined in the Reason Code extension in clause 9.5.3.1.

9.6.2.2 Issuing distribution point extension

This CRL extension identifies the CRL distribution point for public-key certificates for this particular CRL, and indicates if the CRL is indirect, or if it is limited to covering only a subset of the revocation information. If using only partitioned CRLs, the full set of partitioned CRLs shall cover the complete set of public-key certificates whose revocation status will be reported using the CRL mechanism. Thus, the complete set of partitioned CRLs shall be equivalent to a full CRL for the same set of public-key certificates, if the CRL issuer was not using partitioned CRLs. The limitation may be based on a subset of the public-key certificate population or on a subset of revocation reasons. The CRL is signed by the CRL issuer's private key. For a CRL distributed via a directory, the CRL is stored in the entry of the CRL distribution point, which may not be the directory entry of the CRL issuer. If the **issuingDistributionPoint** extension and the **crlScope** extension (deprecated) are both absent, the CRL shall contain entries for all revoked, unexpired public-key certificates.

After a public-key certificate appears on a CRL, it may be deleted from a subsequent CRL after the public-key certificate's expiry. This extension is defined as follows:

```
issuingDistributionPoint EXTENSION ::= {
  SYNTAX          IssuingDistPointSyntax
  IDENTIFIED BY  id-ce-issuingDistributionPoint }

IssuingDistPointSyntax ::= SEQUENCE {
  -- If onlyContainsUserPublicKeyCerts and onlyContainsCACerts are both FALSE,
  -- the CRL covers both certificate types
  distributionPoint               [0]  DistributionPointName OPTIONAL,
  onlyContainsUserPublicKeyCerts  [1]  BOOLEAN DEFAULT FALSE,
  onlyContainsCACerts             [2]  BOOLEAN DEFAULT FALSE,
  onlySomeReasons                 [3]  ReasonFlags OPTIONAL,
  indirectCRL                     [4]  BOOLEAN DEFAULT FALSE,
  onlyContainsAttributeCerts      [5]  BOOLEAN OPTIONAL, -- Use is strongly deprecated
  ... }
```

The **distributionPoint** component contains the name of the distribution point in one or more name forms. If **onlyContainsUserPublicKeyCerts** is **TRUE**, the CRL only contains revocations for end-entity public-key certificates. If **onlyContainsCACerts** is **TRUE**, the CRL only contains revocations for CA certificates. If **onlyContainsUserPublicKeyCerts** and **onlyContainsCACerts** are both **FALSE**, the CRL contains revocations for both end-entity public-key certificates and CA certificates. A CRL shall not contain this extension where both **onlyContainsUserPublicKeyCerts** and **onlyContainsCACerts** are set to **TRUE**. If **onlySomeReasons** is present, the CRL only contains revocations of public-key certificates for the identified reason or reasons; otherwise, the CRL contains revocations for all reasons. If **indirectCRL** is **TRUE**, then the CRL may contain revocation notifications for public-key certificates issued by CAs that have a name different from the name of the issuer of the CRL. The particular CA responsible for each entry is as indicated by the **certificateIssuer** CRL entry extension in that entry or in accordance with the defaulting rules described in clause 9.6.2.3. Consequently, a relying party that is capable of processing a CRL in which **`indirectCRL`** is set to **`TRUE`** shall also be capable of processing the **`certificateIssuer`** CRL entry extension. In such a CRL, it is the responsibility of the CRL issuer to ensure that the CRL is complete in that it contains all revocation entries, consistent with **`onlyContainsUserPublicKeyCerts`**, **`onlyContainsCACerts`**, and **`onlySomeReasons`** indicators, from all authorities that identify this CRL issuer in their public-key certificates.

If **`onlyContainsAttributeCerts`** is **`TRUE`**, the CRL only contains revocations for attribute certificates. This component is deprecated and should not be included. Instead, the **`aAissuingDistributionPoint`** extension should be used.

NOTE 1 – This component was introduced into the fourth edition of this Specification and removed again in the fifth edition. Each of these two actions has caused compatibility problems. This component has been re-introduced into the sixth edition in a way to remove any compatibility issues.

If CRLs are partitioned by reason code, and the reason code changes for a revoked public-key certificate (causing the public-key certificate to move from one CRL stream to another), it is necessary to continue to include the public-key certificate on the CRL stream for the old revocation reason until the **`nextUpdate`** times of all CRLs, that do not list the public-key certificate, on the CRL stream for the new reason code have been reached.

If the CRL contains an **`issuingDistributionPoint`** extension with the **`distributionPoint`** component present, at least one name for the distribution point in the public-key certificate (e.g., **`cRLDistributionPoints`**, **`freshestCRL`**, **`issuer`**) shall match a name for the distribution point in the CRL. Also, it may be the case that only the **`nameRelativeToCRLIssuer`** component is present. In that case, a name comparison would be done on the full distinguished name, constructed by appending the value of the **`nameRelativeToCRLIssuer`** to the distinguished name found in the **`issuer`** component of the CRL. If the names being compared are distinguished names (as opposed to names of other forms within the **`GeneralNames`** construct), the **`distinguishedNameMatch`** matching rule is used to compare the two DNs for equality.

For CRLs distributed via a directory, the following rules apply. If the CRL is a dCRL it shall be distributed via the **`deltaRevocationList`** attribute of the associated distribution point or, if no distribution point is identified, via the **`deltaRevocationList`** attribute of the CRL issuer entry, regardless of the settings for public-key certificate types covered by the CRL. Unless the CRL is a dCRL:

- a CRL which has **`onlyContainsCACerts`** set to **`TRUE`** and does not contain an **`aAissuingDistributionPoint`** extension shall be distributed via the **`authorityRevocationList`** attribute of the associated distribution point or, if no distribution point is identified, via the **`authorityRevocationList`** attribute of the CRL issuer entry;
- a CRL which has **`onlyContainsCACerts`** set to **`TRUE`** and contains an **`aAissuingDistributionPoint`** extension with **`containsUserAttributeCerts`** set to **`FALSE`** shall be distributed via the **`authorityRevocationList`** attribute of the associated distribution point or, if no distribution point is identified, via the **`authorityRevocationList`** attribute of the CRL issuer entry;
- a CRL which has only **`onlyContainsCACerts`** set to **`FALSE`** shall be distributed via the **`certificateRevocationList`** attribute of the associated distribution point or, if no distribution point is identified, via the **`certificateRevocationList`** attribute of the CRL issuer entry.

This extension shall always be flagged as critical. A relying party that does not understand this extension cannot assume that the CRL contains a complete list of revoked public-key certificates of the identified CA. CRLs not containing critical extensions shall contain all current CRL entries for the issuing authority, including entries for all revoked end-entity certificates and authority certificates.

NOTE 2 – The means by which revocation information is communicated by CAs to CRL issuers is beyond the scope of this Specification.

If an authority publishes a CRL with **`onlyContainsUserPublicKeyCerts`** or **`onlyContainsCACerts`** set to **`TRUE`**, then the authority shall ensure that all CA certificates covered by this CRL contain the **`basicConstraints`** extension.

9.6.2.3 Certificate issuer extension

This CRL entry extension identifies the CA associated with an entry in an indirect CRL, i.e., a CRL that has the **`indirectCRL`** indicator set in its issuing distribution point extension. If this extension is not present on the first entry in an indirect CRL, the CA defaults to the CRL issuer. On subsequent entries in an indirect CRL, if this extension is not present, the CA for the entry is the same as that for the preceding entry.

This extension is defined as follows:

```
certificateIssuer EXTENSION ::= {
  SYNTAX          GeneralNames
  IDENTIFIED BY   id-ce-certificateIssuer }
```

This extension shall always be flagged as critical. If an implementation ignores this extension, it cannot correctly pair CRL entries to CAs.

NOTE – This extension may also be included in ACRL entries as specified in clause 17.2.2.3.

9.6.2.4 Delta CRL indicator extension

The delta CRL indicator extension identifies a CRL as being a delta CRL (dCRL) that provides updates to a referenced base CRL. The referenced base CRL is a CRL that was explicitly issued as a CRL that is complete for a given scope. The CRL containing the delta CRL indicator extension contains updates to the public-key certificate revocation status for that same scope. This scope does not necessarily include all revocation reasons or all public-key certificates issued by a CA, especially in the case where the CRL is a CRL distribution point. However, the combination of a CRL containing the delta CRL indicator extension plus the CRL referenced in the **`BaseCRLNumber`** component of this extension is equivalent to a full CRL, for the applicable scope, at the time of publication of the dCRL.

This extension is defined as follows:

```
deltaCRLIndicator EXTENSION ::= {
  SYNTAX         BaseCRLNumber
  IDENTIFIED BY  id-ce-deltaCRLIndicator }

BaseCRLNumber ::= CRLNumber
```

The value of type **`BaseCRLNumber`** identifies the CRL number of the base CRL that was used as the foundation in the generation of this dCRL. The referenced CRL shall be a CRL that is complete for the applicable scope.

This extension shall always be flagged as critical. A relying party that does not understand the use of dCRLs should not use a CRL containing this extension, as the CRL may not be as complete as the user expects.

NOTE – This extension may also be included in dACRL entries as specified in clause 17.2.2.4.

9.6.2.5 Base update time extension

The base update extension is for use in dCRLs and is used to identify the date/time after which this delta provides updates to the revocation status. This extension should only be used in dCRLs that contain the **`deltaCRLIndicator`** extension. A dCRL that instead contains the **`crlScope`** extension (deprecated) does not require this extension as the **`baseThisUpdate`** component of the **`crlScope`** extension can be used for the same purpose.

```
baseUpdateTime EXTENSION ::= {
  SYNTAX         GeneralizedTime
  IDENTIFIED BY  id-ce-baseUpdateTime }
```

This extension shall always be flagged as non-critical.

NOTE – This extension may also be included in dACRL as specified in clause 17.2.2.5.

9.6.2.6 Freshest CRL extension

The freshest CRL extension may be used as either a public-key certificate or CRL extension. Within public-key certificates, this extension may be used in CA certificates, as well as end-entity public-key certificates. This extension identifies the CRL to which a relying party should refer to obtain the freshest revocation information (e.g., latest dCRL). This extension is defined as follows:

```
freshestCRL EXTENSION ::= {
  SYNTAX         CRLDistPointsSyntax
  IDENTIFIED BY  id-ce-freshestCRL }
```

The **`CRLDistPointsSyntax`** data type is as defined in the CRL distribution points extension in clause 9.6.2.1.

This extension may, at the option of the CA, be either be flagged as critical or as non-critical. If the freshest CRL extension is flagged as critical, a relying party shall not use the public-key certificate without first retrieving and checking the freshest CRL. If the extension is flagged as non-critical, the relying party may use local means to determine whether the freshest CRL is required to be checked.

NOTE – This extension may also be included in attribute certificates and ACRLs as specified in clause 17.2.2.6.

10 Delta CRL relationship to base

A dCRL includes a **`deltaCRLIndicator`** extension (or a deprecated **`crlScope`** extension) to indicate the base revocation information that is being updated with this dCRL.

If the **`deltaCRLIndicator`** is present in a dCRL, the base revocation information that is being updated is the base CRL referenced in that extension. The base CRL referenced by a **`deltaCRLIndicator`** extension shall be a CRL that is issued as complete for its scope (i.e., it is not itself a dCRL).

If the deprecated **`crlScope`** extension is present and contains the **`baseRevocationInfo`** component to reference the base revocation information that is being updated, this is a reference to a particular point in time from which this dCRL provides updates. The **`baseRevocationInfo`** component references a CRL that may or may not have been issued as one that is complete for that scope (i.e., the referenced CRL may only have been issued as a dCRL). However, the dCRL containing the **`baseRevocationInfo`** component updates the revocation information that is complete for the scope of the referenced CRL at the time that the referenced CRL was issued. The relying party may apply the dCRL to a CRL that is complete for the given scope and that was issued at the same time as or after the CRL referenced in the dCRL containing the **`baseRevocationInfo`** component was issued.

Because of the potential for conflicting information, a CRL shall not contain both the **`deltaCRLIndicator`** extension and a **`crlScope`** extension with the **`baseRevocationInfo`** component. A CRL may contain both the **`deltaCRLIndicator`** extension and **`crlScope`** extension only if the **`baseRevocationInfo`** component is not present in the **`crlScope`** extension.

A dCRL may also be an indirect CRL in that it may contain updated revocation information related to base CRLs issued by one or more than one CAs (see clause 7.12).

Application of a dCRL to the referenced base revocation information shall accurately reflect the current status of revocation.

– A public-key certificate's revocation notice, with revocation reason **`certificateHold`**, may appear on either a dCRL or a CRL that is complete for a given scope. This reason code is intended to indicate a temporary revocation of the public-key certificate pending a further decision on whether to permanently revoke the public-key certificate or reinstate it as one that is not revoked.

  a) If a public-key certificate was listed as revoked with revocation reason **`certificateHold`** on a CRL (either a dCRL or a CRL that is complete for a given scope), whose **`cRLNumber`** is n, and the hold is subsequently released, the public-key certificate shall be included in all dCRLs issued after the hold is released where the **`cRLNumber`** of the referenced base CRL is less than or equal to n. Depending on the extension used to indicate that this CRL is a dCRL, the CRL number of a referenced base CRL is either the value of the **`BaseCRLNumber`** data type of the **`deltaCRLIndicator`** extension or the **`cRLNumber`** element of the **`BaseRevocationInfo`** data type of the **`cRLScope`** extension. The public-key certificate shall be listed with revocation reason **`removeFromCRL`** unless the public-key certificate is subsequently revoked again for one of the revocation reasons covered by the dCRL, in which case the public-key certificate shall be listed with the revocation reason appropriate for the subsequent revocation.

  b) If the certificate was not removed from hold, but was permanently revoked, then it shall be listed on all subsequent dCRLs where the **`cRLNumber`** of the referenced base CRL is less than the **`cRLNumber`** of the CRL (either a dCRL or a CRL that is complete for the given scope) on which the permanent revocation notice first appeared. Depending on the extension used to indicate that this CRL is a dCRL, the CRL number of a referenced base CRL is either the value of the **`BaseCRLNumber`** data type of the **`deltaCRLIndicator`** extension or the **`cRLNumber`** element of the **`BaseRevocationInfo`** data type of the **`cRLScope`** extension.

– A public-key certificate's revocation notice may first appear on dCRL and it is possible that the public-key certificate's validity period might expire before the next CRL that is complete for the applicable scope is issued. In this situation, that revocation notice shall be included in all subsequent dCRLs until that revocation notice is included on at least one issued CRL that is complete for the scope of that public-key certificate.

A CRL that is complete for a given scope, at the current time, can be constructed locally in either of the following ways:

– by retrieving the current dCRL for that scope, and combining it with an issued CRL that is complete for that scope and that has a **`cRLNumber`** greater than or equal to the **`cRLNumber`** of the base CRL referenced in the dCRL; or

– by retrieving the current dCRL for that scope and combining it with a locally constructed CRL that is complete for that scope and that was constructed with a dCRL that has a **`cRLNumber`** greater than or equal to the **`cRLNumber`** of the base CRL referenced in the current dCRL.

11 Authorization and validation lists

11.1 Authorization and validation list concept

An authorization and validation list (AVL) is a list providing authorization and validation information to a relying party, called an AVL entity. Such an AVL entity may also act as an end entity been equipped with an end-entity public-key certificate. An AVL provides information to an AVL entity about potential communications partner and possible restrictions on how and what to communicate with such partners.

In some environments, entities may have a requirement only to communicate with a few selected entities and may not accept PDUs from any other entity. Validation may be optimized in such environments by use of AVLs. If a PDU is received from an entity not represented in the AVL or the constraints on the communication are violated, the PDU may be discarded, subject to local policies. An AVL is created, maintained and signed by an entity called the authorizer, which is an outside entity to which responsibility for all or part of public-key certificate validation has been outsourced. An authorizer is typically closely related to the AVL entities it serves, e.g., being part of the same organization. It may then act as a trust anchor for the AVL entities it serves. These AVL entities may hold trust anchor information of the authorizer. For some AVL entities, this may be the only trust anchor information needed.

Three scenarios are recognized here:

a) An AVL is placed in an AVL entity with no or minor constraints allowing the AVL entity to perform much of the validation on its own. In this case, the AVL is mainly used to restrict communications to selected entities and possibly to put restriction on such communications. An AVL may in this scenario have information representing each public-key certificates allocated to potential communication partners and/or it may have information representing a group of communication partners having the same distinguished naming prefix.

b) An AVL is placed in an AVL entity that is constrained with respect to processing power, storage, communications capabilities and/or response time to a degree where it cannot afford to go to a third party when doing validation, meaning that the entity locally needs to have sufficient updated information available to do the validation. In this scenario, the authorizer is using a subscription service to get revocation information from relevant CAs and updates the AVLs at AVL entities as required.

c) An AVL entity may not have continuously online access to revocation information, such as OCSP, and at the same time not be able to hold long CRLs. In this case, it is also relevant to have reasonable updated validation information available locally. Also in this scenario, the authorizer is using a subscription service to get revocation information from relevant CAs and updates the AVLs at AVL entities as required when communication is possible.

d) A relying party, not being an AVL entity, may not have one continuously online access to revocation information. Such a relying party does not use the service of an authorizer, but uses the same subscription type as an authorizer to keep status information about a set public-key certificates for potential communication partners.

11.2 The authorizer

The behaviour of the authorizer is dependent on whether it maintains AVLs for AVL entities in a non-constrained environment or for AVL entities in a constrained environment.

If the AVL is to be placed in a non-constrained AVL entity, the AVL shall only contain rather stable information not affected by state changes of associated public-key certificates.

If the AVL is to be placed in a constrained AVL entity, the authorizer shall locally maintain a list of complete certification paths required for each of the peer entities with which the AVL entity communicates. The authorizer shall continuously ensure that the certification paths validate positively. This included checking for restrictions and expired or revoked public-key certificates. When a certification path can no longer validate positively, affected AVLs may be renewed at affected AVL entities. It is a local policy as to whether or when such an update occurs.

NOTE – An invalid certification path may cause critical entities to be put out of operation. It is outside the scope of this Specification to provide recommendations as to how or when a certification path is failing to a degree where an affected AVL entity should stop communication with some other entity.

11.3 Authorization and validation list syntax

The **`CertAVL`** ASN.1 data type specifies the syntax of an AVL.

```
CertAVL ::= SIGNED {TBSCertAVL}
```

```
TBSCertAVL ::= SEQUENCE {
  version                 [0]  IMPLICIT Version DEFAULT v1,
  serialNumber                 AvlSerialNumber OPTIONAL,
  signature                    AlgorithmIdentifier {{SupportedAlgorithms}},
  issuer                       Name,
  constrained                  BOOLEAN,
  entries                      SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    idType                       CHOICE {
      certIdentifier        [0]  PKCertIdentifier,
      entityGroup                  DistinguishedName, -- only for constrained = FALSE
      ... },
    scope                 [0]  IMPLICIT ScopeRestrictions OPTIONAL,
    entryExtensions       [1]  IMPLICIT Extensions OPTIONAL,
    ... },
  ...,
  ...,
  avlExtensions                Extensions OPTIONAL }

AvlSerialNumber ::= INTEGER (0..MAX)

PKCertIdentifier ::= CHOICE {
  issuerSerialNumber           IssuerSerialNumber,
  fingerprintPKC          [0]  IMPLICIT FINGERPRINT {Certificate},
  fingerprintPK           [1]  IMPLICIT FINGERPRINT {PublicKey},
  ... }

IssuerSerialNumber ::= SEQUENCE {
  issuer          Name,
  serialNumber    CertificateSerialNumber,
  ... }

ScopeRestrictions ::= SEQUENCE OF ScopeRestriction

SCOPE-RESTRICTION ::= TYPE-IDENTIFIER

ScopeRestriction ::= SEQUENCE {
  id              SCOPE-RESTRICTION.&id,
  restriction     SCOPE-RESTRICTION.&Type,
  ... }
```

The **TBSCertAVL** components are specified in the following.

The **version** component shall hold the version of the AVL. This component shall either be absent or have the value **v1**.

The **serialNumber** component, when present, shall contain a serial number that is unique for a specific authorizer within the AVL entity where the AVL is placed. This component shall be present if the authorizer places more than one AVL in a particular entity. Otherwise, it is optional.

NOTE 1 – Multiple authorizers may in principle each place multiple AVLs in a particular entity. This capability is included to cope with more complex environments and should be used with great care. In a typical environment, only a single AVL from a single authorizer will be place in a particular AVL entity.

The **signature** component shall contain the algorithm identifier for the signature algorithm used by the authorizer when signing the AVL. This component shall have the same value as used in the **algorithmIdentifier** component of the **SIGNATURE** data type when signing the AVL.

NOTE 2 – By including this component, the signature algorithm is protected by the signature on the AVL.

The **issuer** component shall be the distinguished name of the authorizer issuing the AVL. It shall be the same name as the subject name of the public-key certificate used for evaluating the signature on the AVL.

The **constrained** component shall take the value **TRUE** if the AVL is to be placed in a constrained AVL entity, and it shall take the value **FALSE** if the AVL is to be placed in a non-constrained AVL entity.

The **entries** component shall consist of a number of elements each representing an AVL entry. Each element shall have the following components:

a) The **idType** component has the following two alternatives:

   – The **certIdentifier** alternative shall hold an identifier of the public-key certificate represented by this element. This alternative shall always be taken if the **constrained** component takes the value **TRUE**. It has the following three alternatives:

i) the **issuerSerialnumber** alternative, which is a sequence of the distinguished name of the public-key certificate issuer and the public-key certificate serial number;

ii) the **fingerprintPKC** alternative, which is a fingerprint of the public-key certificate; or

iii) the **fingerprintPK** alternative, which is a fingerprint of the public key within the public-key certificate.

– The **entityGroup** alternative shall be taken when the entry represents an entity group with which the entity may accept communications. An entity group is characterized by having a distinguished name prefix in common within the name of the **subject** component for public-key certificates issued to the members of the group. This alternative shall not be taken if the **constrained** component takes the value **TRUE**.

b) The **scope** component, when present, specifies one or more restrictions to be observed by the entity. A scope restriction is identified by an object identifier. The **SCOPE-RESTRICTION** information object class is used to bind the syntax of the restriction to an identifying object identifier. A defined restriction shall state to what degree a violation of the restriction will cause a received PDU to be discarded. This Specification defines some general usable restrictions in clause 11.4. Other organizations may define additional restrictions relevant for their organizations.

c) The **entryExtensions** component, when present, shall contain one or more AVL entry extensions.

The **avlExtensions** component, when present, shall contain one or more AVL extensions.

11.4 Authorization and validation restrictions

11.4.3 Protocol restrictions

The **protRestrict** restriction may be added to an AVL entry to limit the protocols acceptable to the AVL entity holding the AVL.

```
protRestrict SCOPE-RESTRICTION ::= {
              ProtRestriction
IDENTIFIED BY id-protRestrict }

ProtRestriction ::= SEQUENCE (SIZE (1..MAX)) OF OBJECT IDENTIFIER
```

A value of the **ProtRestriction** data type shall hold one or more object identifiers identifying communications protocols. When a peer entity attempts to establish a communication with an AVL entity having an AVL with this restriction using a protocol not reflected by a listed object identifier, the communications shall be rejected.

12 Certification path processing procedure

Certification path processing is carried out by a relying party, a trust broker or authorizer, which needs to verify that a particular end-entity public-key certificate is trustworthy. The extensions for certificate policies, basic constraints, name constraints and policy constraints have been designed to facilitate automated, self-contained implementation of certification path processing logic.

The following is an outline of a procedure for validating certification paths. An implementation shall be functionally equivalent to the external behaviour resulting from this procedure. The algorithm used by a particular implementation to derive the correct output(s) from the given inputs is not standardized.

12.1 Path processing inputs

The inputs to the certification path processing procedure are:

a) a set of public-key certificates comprising a certification path;

b) trust anchor information with a public-key for use in verifying the first public-key certificate in the certification path;

c) an initial-policy-set comprising one or more certificate policy identifiers, indicating that any one of these policies would be acceptable to the relying party for the purposes of certification path processing; this input can also take the special value any-policy, but it cannot be null;

d) an initial-explicit-policy indicator value, which indicates if an acceptable policy identifier needs to explicitly appear in the certificate policies extension of all public-key certificates in the path;

e) an initial-policy-mapping-inhibit indicator value, which indicates if policy mapping is forbidden in the certification path;

f) an initial-inhibit-policy indicator value, which indicates if the special value **`anyPolicy`**, if present in a certificate policies extension, is considered a match for any specific certificate policy value in a constrained set;

g) the current date/time (if not available internally to the relying party);

h) an initial-permitted-subtrees-set containing an initial set of subtree specifications defining subtrees within which subject names (of the name form used to specify the subtrees) are permitted. In the public-key certificates in the certification path all subject names of a given name form, for which initial permitted subtrees are defined, shall fall within the permitted subtrees set for that given name form. This input may also contain the special value unbounded to indicate that initially all subject names are acceptable. Subject names are those name values appearing in the **`subject`** component or the **`subjectAltName`** extension;

i) an initial-excluded-subtrees-set containing an initial set of subtree specifications defining subtrees within which the subject names in the public-key certificates in the certification path cannot fall. This input may also be an empty set to indicate that initially no subtree exclusions are in effect;

j) an initial-required-name-forms containing an initial set of name forms indicating that all public-key certificates in the path must include a subject name of at least one of the specified name forms. This input may also be an empty set to indicate that no specific name forms are required for subject names in the certificates.

The values of c), d), e) and f) will depend upon the policy requirements of the user-application combination that needs to use the certified end-entity public key.

Because these are individual inputs to the path validation process, a relying party may limit the trust it places in any given trusted public key to a given set of certificate policies. This can be achieved by ensuring that a given public key is the input to the process only when initial-policy-set input includes policies for which the relying party trusts that public key. Since another input to the process is the certification path itself, this control could be exercised on a transaction by transaction basis.

12.2 Path processing outputs

The outputs of the procedure are:

a) an indication of success or failure of certification path validation;

b) if validation failed, a diagnostic code indicating the reason for failure;

c) the set of authorities-constrained policies and their associated qualifiers in accordance with which the certification path is valid, or the special value any-policy;

d) the set of user-constrained policies, formed from the intersection of the authorities-constrained-policy-set and the initial-policy-set;

e) explicit-policy-indicator, indicating whether the relying party or an authority in the path requires that an acceptable policy be identified in every public-key certificate in the path; and

f) details of any policy mapping that occurred in processing the certification path;

g) if the evaluation was performed by a trust broker, the requesting entity is informed about the outcome (see clause 22);

h) if the evaluation was performed by an authorizer, then:

– if the authorizer is using the information for generating an AVL, then:

i) if the procedure yielded success, the appropriate entry may be added to the AVL

ii) if the procedure yielded failure, depending on type of failure, it is a local policy issue whether to add the entry to the AVL or to leave it out,

– if the authorizer is using the information for possible update of one or more AVLs, then:

i) if the procedure yielded success, the affected AVLs do not need to be replaced based on the validation of the public-key certificate in question

ii) if the procedure yielded failure, depending on type of failure, it is a local policy issue whether to replace affected AVLs.

NOTE – If validation is successful, the relying party may still choose not to use the public-key certificate as a result of values of policy qualifiers or other information in the public-key certificate.

12.3 Path processing variables

The procedure makes use of the following set of state variables:

a) authorities-constrained-policy-set: A table of policy identifiers and qualifiers from the public-key certificates of the certification path (rows represent policies, their qualifiers and mapping history, and columns represent public-key certificates in the certification path).

b) permitted-subtrees: A set of subtree specifications defining subtrees within which all subject names in subsequent public-key certificates in the certification path need to fall, or may take the special value unbounded.

c) excluded-subtrees: A (possibly empty) set of subtree specifications (each comprising a subtree base name and maximum and minimum level indicators) defining subtrees within which no subject name in a subsequent public-key certificate in the certification path may fall.

d) required-name-forms: A (possibly empty) set of sets of name forms. For each set of name forms, every subsequent public-key certificate must contain a name of one of the name forms in the set.

e) explicit-policy-indicator: Indicates whether an acceptable policy needs to be explicitly identified in every public-key certificate in the path.

f) path depth: An integer equal to one more than the number of public-key certificates in the certification path for which processing has been completed.

g) policy-mapping-inhibit-indicator: Indicates whether policy mapping is inhibited.

h) inhibit-any-policy-indicator: Indicates whether the special value **`anyPolicy`** is considered a match for any specific certificate policy.

i) pending-constraints: Details of explicit-policy inhibit-policy-mapping and/or inhibit-any-policy constraints which have been stipulated but have yet to take effect. There are three one-bit indicators called explicit-policy-pending, policy-mapping-inhibit-pending and inhibit-any-policy-pending together with, for each, an integer called skip-certificates which gives the number of public-key certificates yet to skip before the constraint takes effect.

12.4 Initialization step

The procedure involves an initialization step, followed by a series of public-key certificate-processing steps. The initialization step comprises:

1) Write any-policy in the zeroth and first columns of the zeroth row of the authorities-constrained-policy-set table.

2) Initialize the permitted-subtrees variable to the initial-permitted-subtrees-set value.

3) Initialize the excluded-subtrees variable to the initial-excluded-subtrees-set value.

4) Initialize the required-name-forms variable to the initial-required-name-forms value.

5) Initialize the explicit-policy-indicator to the initial-explicit-policy value.

6) Initialize path-depth to one.

7) Initialize the policy-mapping-inhibit-indicator to the initial-policy-mapping-inhibit value.

8) Initialize the inhibit-any-policy-indicator to the initial-inhibit-policy value.

9) Initialize the three pending-constraints indicators to unset.

12.5 Public-key certificate processing

Each public-key certificate is then processed in turn, starting with the public-key certificate signed using the trust anchor public key. The last public-key certificate is considered to be the end-entity public-key certificate; any other public-key certificates are considered to be intermediate CA certificates.

12.5.1 Basic public-key certificate checks

The following checks are applied to a public-key certificate. Self-signed certificates, if encountered in the path, are ignored.

a) Check that the signature verifies, that dates are valid, that the public-key certificate subject and public-key certificate issuer names chain correctly, and that the public-key certificate has not been revoked.

b) For an intermediate version 3 CA certificate, check that **`basicConstraints`** is present and that the **`cA`** component in the **`basicConstraints`** extension is **`TRUE`**. If the **`pathLenConstraint`** component is present, check that the current certification path does not violate that constraint (ignoring intermediate self-issued certificates).

c) If the public-key certificate policies extension is not present, then set the authorities-constrained-policy-set to null by deleting all rows from the authorities-constrained-policy-set table.

d) If the public-key certificate policies extension is present, then for each policy, P, in the extension other than **`anyPolicy`**, attach the policy qualifiers associated with P to each row in the authorities-constrained-policy-set table whose [path-depth] column entry contains the value P. If no row in the authorities-constrained-policy-set table contains P in its [path-depth] column entry but the value in authorities-constrained-policy-set[0, path-depth] is any-policy, then add a new row to the table by duplicating the zeroth row and writing the policy identifier P along with its qualifiers in the [path-depth] column entry of the new row.

e) If the public-key certificate policies extension is present and does not include the value **`anyPolicy`** or if the inhibit-any-policy-indicator is set and the certificate is not a self-issued intermediate certificate, then delete any row for which the [path-depth] column entry contains the value any-policy along with any row for which the [path-depth] column entry does not contain one of the values in the certificate policies extension.

f) If the public-key certificate policies extension is present and includes the value **`anyPolicy`** and the inhibit-any-policy-indicator is not set, then attach the policy qualifiers associated with **`anyPolicy`** to each row in the authorities-constrained-policy-set table whose [path-depth] column entry contains the value any-policy or contains a value that does not appear in the certificate policies extension.

g) If the public-key certificate is not an intermediate self-issued certificate, check that the subject name is within the name-space given by the value of permitted-subtrees and is not within the name-space given by the value of excluded-subtrees.

h) If the public-key certificate is not an intermediate self-issued certificate, and if required-name-forms is not an empty set, for each set of name forms in required-name-forms check that there is a subject name in the public-key certificate of one of the name forms in the set.

12.5.2 Processing intermediate certificates

For an intermediate CA certificate, the following constraint recording actions are then performed, in order to correctly set up the state variables for the processing of the next certificate. Self-signed certificates, if encountered in the path, are ignored.

1) If the **`nameConstraints`** extension with a **`permittedSubtrees`** component is present in the certificate, set the permitted-subtrees state variable to the intersection of its previous value and the value indicated in the certificate extension.

2) If the **`nameConstraints`** extension with an **`excludedSubtrees`** component is present in the certificate, set the excluded-subtrees state variable to the union of its previous value and the value indicated in the certificate extension.

3) If policy-mapping-inhibit-indicator is set:
   – process any policy mappings extension by, for each mapping identified in the extension, locating all rows in the authorities-constrained-policy-set table whose [path-depth] column entry is equal to the issuer domain policy value in the extension and delete the row.

4) If policy-mapping-inhibit-indicator is not set:
   – process any policy mappings extension by, for each mapping identified in the extension, locating all rows in the authorities-constrained-policy-set table whose [path-depth] column entry is equal to the issuer domain policy value in the extension, and write the subject domain policy value from the extension in the [path-depth+1] column entry of the same row. If the extension maps an issuer domain policy to more than one subject domain policy, then the affected row is copied and the new entry added to each row. If the value in authorities-constrained-policy-set[0, path-depth] is any-policy, then write each issuer domain policy identifier from the policy mappings extension in the [path-depth] column, making duplicate rows as necessary and retaining qualifiers if they are present, and write the subject domain policy value from the extension in the [path-depth+1] column entry of the same row;
   – if the policy-mapping-inhibit-pending indicator is set and the certificate is not self-issued, decrement the corresponding skip-certificates value and, if this value becomes zero, set the policy-mapping-inhibit-indicator;

  - if the `inhibitPolicyMapping` component of the `policyConstraints` extension is present in the certificate, perform the following. For a `SkipCerts` value of 0, set the policy-mapping-inhibit-indicator. For any other `SkipCerts` value, set the policy-mapping-inhibit-pending indicator, and set the corresponding skip-certificates value to the lesser of the `SkipCerts` value and the previous skip-certificates value (if the policy-mapping-inhibit-pending indicator was already set).

5) For any row not modified in step 3) (and every row in the case that there is no mapping extension present in the certificate), write the policy identifier from [path-depth] column in the [path-depth+1] column of the row.

6) If inhibit-any-policy-indicator is not set:

  - If the inhibit-any-policy-pending indicator is set and the certificate is not self-issued, decrement the corresponding skip-certificates value and, if this value becomes zero, set the inhibit-any-policy-indicator.
  - If the `inhibitAnyPolicy extension` is present in the certificate, perform the following. For a `SkipCerts` value of 0, set the inhibit-any-policy-indicator. For any other `SkipCerts` value, set the inhibit-any-policy-pending indicator, and set the corresponding skip-certificates value to the lesser of the `SkipCerts` value and the previous skip-certificates value (if the inhibit-any-policy-pending indicator was already set).

7) Increment [path-depth].

12.5.3 Explicit policy indicator processing

For all certificates, the following actions are then performed:

1) If explicit-policy-indicator is not set:

  - if the explicit-policy-pending indicator is set and the certificate is not a self-issued intermediate CA certificate, decrement the corresponding skip-certificates value and, if this value becomes zero, set explicit-policy-indicator.
  - If the `requireExplicitPolicy component of the policyConstraints` extension is present in the certificate, perform the following. For a `SkipCerts` value of 0, set the explicit-policy-indicator. For any other `SkipCerts` value, set the explicit-policy-pending indicator, and set the corresponding skip-certificates value to the lesser of the `SkipCerts` value and the previous skip-certificates value (if the explicit-policy-pending indicator was already set).
  - If the `requireExplicitPolicy` component of the `policyConstraints` is present, and the certification path includes a certificate issued by a nominated CA, it is necessary for all certificates in the path to contain, in the certificate policies extension, an acceptable policy identifier. An acceptable policy identifier is the identifier of the certificate policy required by the user of the certification path, the identifier of a policy which has been declared equivalent to it through policy mapping, or any-policy. The nominated CA is either the issuer CA of the certificate containing this extension (if the value of `requireExplicitPolicy` is 0) or a CA which is the subject of a subsequent certificate in the certification path (as indicated by a non-zero value).

12.5.4 Final processing

Once all certificates in the path have been processed, the following actions are then performed:

1) Determine the authorities-constrained-policy-set from the authorities-constrained-policy-set table. If the table is empty, then the authorities-constrained-policy-set is the empty or null set. If the authorities-constrained-policy-set[0, path-depth] is any-policy, then the authorities-constrained-policy-set is any-policy. Otherwise, the authorities-constrained-policy-set is, for each row in the table, the value in the left-most cell which does not contain the identifier any-policy.

2) Calculate the user-constrained-policy-set by forming the intersection of the authorities-constrained-policy-set and the initial-policy-set.

3) If the explicit-policy-indicator is set, check that neither the authorities-constrained-policy-set nor the user-constrained-policy-set is empty.

If any of the above checks were to fail, then the procedure shall terminate, returning a failure indication, an appropriate reason code, the *explicit-policy-indicator*, the *authorities-constrained-policy-set* and the *user-constrained-policy-set*. If the failure is due to an empty *user-constrained-policy-set*, then the path is valid under the authority-constrained policy(s), but none is acceptable to the user.

If none of the above checks were to fail on the end certificate, then the procedure shall terminate, returning a success indication together with the *explicit-policy-indicator*, the *authorities-constrained-policy-set* and the *user-constrained-policy-set*.

13 PKI directory schema

This clause defines the directory schema elements used to represent PKI information in a directory. It includes specifications of relevant object classes, attribute types and attribute value matching rules.

13.1 PKI directory object classes and name forms

This subclause includes the definition of object classes used to represent PKI objects in a directory.

13.1.1 PKI user object class

The PKI user object class is used in defining entries for objects that may be the subject of end-entity public-key certificates.

```
pkiUser OBJECT-CLASS ::= {
  SUBCLASS OF          {top}
  KIND                 auxiliary
  MAY CONTAIN          {userCertificate}
  LDAP-NAME            {"pkiUser"}
  LDAP-DESC            "X.509 PKI User"
  ID                   id-oc-pkiUser }
```

13.1.2 PKI CA object class

The PKI CA object class is used in defining entries for objects (entities) that act as CAs.

```
pkiCA OBJECT-CLASS ::= {
  SUBCLASS OF          {top}
  KIND                 auxiliary
  MAY CONTAIN          {cACertificate |
                        certificateRevocationList |
                        eepkCertificateRevocationList |
                        authorityRevocationList |
                        crossCertificatePair}
  LDAP-NAME            {"pkiCA"}
  LDAP-DESC            "X.509 PKI Certificate Authority"
  ID                   id-oc-pkiCA }
```

13.1.3 CRL distribution points object class and name form

The CRL distribution point object class is used in defining entries for objects which act as CRL distribution points.

```
cRLDistributionPoint OBJECT-CLASS ::= {
  SUBCLASS OF          {top}
  KIND                 structural
  MUST CONTAIN         {commonName}
  MAY CONTAIN          {certificateRevocationList |
                        eepkCertificateRevocationList |
                        authorityRevocationList |
                        deltaRevocationList}
  LDAP-NAME            {"cRLDistributionPoint"}
  LDAP-DESC            "X.509 CRL distribution point"
  ID                   id-oc-cRLDistributionPoint }
```

The CRL distribution point name form specifies how entries of object class `cRLDistributionPoint` may be named.

```
cRLDistPtNameForm NAME-FORM ::= {
  NAMES                cRLDistributionPoint
  WITH ATTRIBUTES      {commonName}
  ID                   id-nf-cRLDistPtNameForm }
```

13.1.4 Delta CRL object class

The delta CRL object class is used in defining entries for objects that hold delta revocation lists (e.g., CAs, AAs etc.).

```
deltaCRL OBJECT-CLASS ::= {
  SUBCLASS OF          {top}
  KIND                auxiliary
  MAY CONTAIN         {deltaRevocationList}
  LDAP-NAME           {"deltaCRL"}
  LDAP-DESC           "X.509 delta CRL"
  ID                  id-oc-deltaCRL }
```

13.1.5 Certificate Policy and CPS object class

The CP CPS object class is used in defining entries for objects that contain certificate policy and/or certification practice information.

```
cpCps OBJECT-CLASS ::= {
  SUBCLASS OF         {top}
  KIND                auxiliary
  MAY CONTAIN         {certificatePolicy |
                       certificationPracticeStmt}
  LDAP-NAME           {"cpCps"}
  LDAP-DESC           "Certificate Policy and Certification Practice Statement"
  ID                  id-oc-cpCps }
```

13.1.6 PKI certification path object class

The PKI cert path object class is used in defining entries for objects that contain PKI paths. It will generally be used in conjunction with entries that include auxiliary object class **pkiCA** or **pkiUser**.

```
pkiCertPath OBJECT-CLASS ::= {
  SUBCLASS OF         {top}
  KIND                auxiliary
  MAY CONTAIN         {pkiPath}
  LDAP-NAME           {"pkiCertPath"}
  LDAP-DESC           "PKI Certification Path"
  ID                  id-oc-pkiCertPath }
```

13.2 PKI directory attributes

This subclause includes the definition of directory attributes to store PKI information elements in a directory.

13.2.1 User certificate attribute

An attribute value of type **userCertificate** holds an end-entity public-key certificate.

An entity may obtain one or more end-entity public-key certificates from one or more CAs.

```
userCertificate ATTRIBUTE ::= {
  WITH SYNTAX              Certificate
  EQUALITY MATCHING RULE   certificateExactMatch
  LDAP-SYNTAX              x509Certificate.&id
  LDAP-NAME                {"userCertificate"}
  LDAP-DESC                "X.509 user certificate"
  ID                       id-at-userCertificate }
```

13.2.2 CA certificate attribute

The **cACertificate** attribute of a CA's directory entry shall be used to store self-issued certificates (if any) and CA certificates issued to this CA by CAs in the same realm as this CA. The definition of realm is purely a matter of local policy.

```
cACertificate ATTRIBUTE ::= {
  WITH SYNTAX              Certificate
  EQUALITY MATCHING RULE   certificateExactMatch
  LDAP-SYNTAX              x509Certificate.&id
  LDAP-NAME                {"cACertificate"}
  LDAP-DESC                "X.509 CA certificate"
  ID                       id-at-cAcertificate }
```

13.2.3 Cross-certificate pair attribute type

The **issuedToThisCA** component of the **crossCertificatePair** attribute of a CA's directory entry shall be used to store all, except self-issued certificates issued to this CA. Optionally, the **issuedByThisCA** component of the **crossCertificatePair** attribute, of a CA's directory entry may contain a subset of CA certificates issued by this CA to other CAs. If a CA issues a certificate to another CA, and the subject CA is not a subordinate to the issuer CA in a hierarchy, then the issuer CA shall place that certificate in the **issuedByThisCA** element of the **crossCertificatePair** attribute of its own directory entry. When both the **issuedToThisCA** and the **issuedByThisCA** elements are present in a single attribute value, the issuer name in one CA certificate shall match the subject name in the other and vice versa, and the subject public key in one CA certificate shall be capable of verifying the digital signature on the other CA certificate and vice versa. The term **forward** was used in previous editions for **issuedToThisCA** and the term **reverse** was used in previous editions for **issuedByThisCA**.

When an **issuedByThisCA** element is present, the **issuedToThisCA** element value and the **issuedByThisCA** element value need not be stored in the same attribute value; in other words, they can be stored in either a single attribute value or two attribute values.

```
crossCertificatePair ATTRIBUTE ::= {
  WITH SYNTAX                CertificatePair
  EQUALITY MATCHING RULE     certificatePairExactMatch
  LDAP-SYNTAX                x509CertificatePair.&id
  LDAP-NAME                  {"crossCertificatePair"}
  LDAP-DESC                  "X.509 cross certificate pair"
  ID                         id-at-crossCertificatePair }

CertificatePair ::= SEQUENCE {
  issuedToThisCA   [0]  Certificate OPTIONAL,
  issuedByThisCA   [1]  Certificate OPTIONAL,
  ... }
  (WITH COMPONENTS { ..., issuedToThisCA PRESENT} |
   WITH COMPONENTS { ..., issuedByThisCA PRESENT})
```

13.2.4 Public-key certificate revocation list attribute type

A value of this attribute type holds a CRL providing information about revoked public-key certificates.

```
certificateRevocationList ATTRIBUTE ::= {
  WITH SYNTAX                CertificateList
  EQUALITY MATCHING RULE     certificateListExactMatch
  LDAP-SYNTAX                x509CertificateList.&id
  LDAP-NAME                  {"certificateRevocationList"}
  LDAP-DESC                  "X.509 certificate revocation list"
  ID                         id-at-certificateRevocationList }
```

13.2.5 End-entity public-key certificate revocation list attribute type

A value of the following attribute type holds a CRL providing information about revoked end-entity public-key certificates.

```
eepkCertificateRevocationList ATTRIBUTE ::= {
  WITH SYNTAX                CertificateList
  EQUALITY MATCHING RULE     certificateListExactMatch
  LDAP-SYNTAX                x509CertificateList.&id
  LDAP-NAME                  {"eepkCertificateRevocationList"}
  LDAP-DESC                  "X.509 EEPK certificate revocation list"
  ID                         id-at-eepkCertificateRevocationList }
```

13.2.6 CA revocation list attribute type

A value of the following attribute type contains a list of revoked CA certificates.

```
authorityRevocationList ATTRIBUTE ::= {
  WITH SYNTAX                CertificateList
  EQUALITY MATCHING RULE     certificateListExactMatch
  LDAP-SYNTAX                x509CertificateList.&id
  LDAP-NAME                  {"authorityRevocationList"}
  LDAP-DESC                  "X.509 CA revocation list"
  ID                         id-at-authorityRevocationList }
```

13.2.7 Delta revocation list attribute

The following attribute type is defined for holding a dCRL in a directory entry:

```
deltaRevocationList ATTRIBUTE ::= {
  WITH SYNTAX                CertificateList
  EQUALITY MATCHING RULE     certificateListExactMatch
  LDAP-SYNTAX                x509CertificateList.&id
  LDAP-NAME                  {"deltaRevocationList"}
  LDAP-DESC                  "X.509 delta revocation list"
  ID                         id-at-deltaRevocationList }
```

13.2.7 Supported algorithms attribute

A directory attribute is defined to support the selection of an algorithm for use when communicating with a remote end entity using end-entity public-key certificates as defined in this Specification. The following ASN.1 defines this (multi-valued) attribute type:

```
supportedAlgorithms ATTRIBUTE ::= {
  WITH SYNTAX                SupportedAlgorithm
  EQUALITY MATCHING RULE     algorithmIdentifierMatch
  LDAP-SYNTAX                x509SupportedAlgorithm.&id
  LDAP-NAME                  {"supportedAlgorithms"}
  LDAP-DESC                  "X.509 support algorithms"
  ID                         id-at-supportedAlgorithms }

SupportedAlgorithm ::= SEQUENCE {
  algorithmIdentifier                AlgorithmIdentifier{{SupportedAlgorithms}},
  intendedUsage                 [0]  KeyUsage OPTIONAL,
  intendedCertificatePolicies   [1]  CertificatePoliciesSyntax OPTIONAL,
  ... }
```

Each value of the multi-valued attribute shall have a distinct **algorithmIdentifier** value. The value of the **intendedUsage** component provides an indication of the intended usage of the algorithm (see clause 9.2.2.3 for recognized uses). The value of the **intendedCertificatePolicies** component identifies the certificate policies and, optionally, certificate policy qualifiers with which the identified algorithm may be used.

13.2.8 Certification practice statement attribute

The **certificationPracticeStmt** attribute is used to store information about a CA's certification practice statement.

```
certificationPracticeStmt ATTRIBUTE ::= {
  WITH SYNTAX  InfoSyntax
  ID           id-at-certificationPracticeStmt }

InfoSyntax ::= CHOICE {
  content  UnboundedDirectoryString,
  pointer  SEQUENCE {
    name     GeneralNames,
    hash     HASH{HashedPolicyInfo} OPTIONAL,
    ... },
  ... }

POLICY ::= TYPE-IDENTIFIER

HashedPolicyInfo ::= POLICY.&Type({Policies})

Policies POLICY ::= {...} -- Defined by implementers
```

If **content** is present, the complete content of the authority's certification practice statement is included.

If **pointer** is present, the **name** component references one or more locations where a copy of the authority's certification practice statement can be located. If the **hash** component is present, it contains a HASH of the content of the certification practice statement that should be found at a referenced location. This hash can be used to perform an integrity check of the referenced document.

13.2.9 Certificate policy attribute

The **certificatePolicy** attribute is used to store information about a certificate policy.

```
certificatePolicy ATTRIBUTE ::= {
  WITH SYNTAX  PolicySyntax
  ID           id-at-certificatePolicy }

PolicySyntax ::= SEQUENCE {
  policyIdentifier  PolicyID,
  policySyntax      InfoSyntax,
  ... }

PolicyID ::= CertPolicyId
```

The **policyIdentifier** component includes the object identifier registered for the particular certificate policy.

The **policySyntax** component (see clause 13.2.8) has the following two alternatives:

a) If the **content** alternative is taken, the complete content of the certificate policy is included.

b) If the **pointer** alternative is taken, then

   – the **name** component references one or more locations where a copy of the certificate policy can be located.

   – If the **hash** component is present, it contains a HASH of the content of the certificate policy that should be found at a referenced location. This hash can be used to perform an integrity check of the referenced document.

   NOTE – The option to include a hash in this attribute is purely to perform an integrity check against data located from a source other than a directory. The HASH stored in a directory needs to be protected. Directory security services, including strong authentication, access control and/or signed attributes could be used for this purpose. In addition, even if the HASH matches the original CP/CPS document, there are additional security requirements to ensure that the original specification itself is the correct document (e.g., the document is signed by an appropriate authority).

13.2.10 PKI path attribute

The PKI path attribute is used to store certification paths, each consisting of a sequence of public-key certificates.

```
pkiPath ATTRIBUTE ::= {
  WITH SYNTAX  PkiPath
  ID           id-at-pkiPath }
```

An attribute of this type may be stored in a directory entry of object class **pkiCA** or **pkiUser**.

When stored in **pkiCA** entries, values of this attribute type contain certification paths excluding end-entity public-key certificates. As such, the attribute is used to store certification paths that are frequently used by relying parties associated with that CA. A value of this attribute can be used in conjunction with any end-entity public-key certificate issued by the last certificate subject in the attribute value.

When stored in **pkiUser** entries, values of this attribute type contain certification paths that include the end-entity public-key certificate. In this case, the end entity is the entity whose entry holds this attribute. The values of the attribute represent complete certification paths for public-key certificates issued to this user.

13.3 PKI directory matching rules

This subclause defines matching rules for use with values of attribute types with syntax **Certificate**, **CertificatePair**, **CertificateList**, **CertificatePolicy**, and **SupportedAlgorithm**, respectively. This subclause also defines matching rules to facilitate the selection of ´public-key certificates or CRLs with specific characteristics from multi-valued attributes holding multiple public-key certificates or CRLs. The enhanced certificate matching rule provides the ability to perform more sophisticated matching against public-key certificates held in directory entries.

13.3.1 Certificate exact match

The certificate exact match rule compares for equality a presented value with an attribute value with syntax **Certificate**. It uniquely selects a single certificate.

```
certificateExactMatch MATCHING-RULE ::= {
  SYNTAX  CertificateExactAssertion
  ID      id-mr-certificateExactMatch }

CertificateExactAssertion ::= SEQUENCE {
  serialNumber  CertificateSerialNumber,
  issuer        Name,
  ... }
```

This matching rule returns TRUE if the components in the attribute value match those in the presented value.

13.3.2 Certificate match

The certificate match rule compares a presented value with an attribute value with syntax **Certificate**. It selects one or more certificates on the basis of various characteristics.

```
certificateMatch MATCHING-RULE ::= {
  SYNTAX  CertificateAssertion
  ID      id-mr-certificateMatch }

CertificateAssertion ::= SEQUENCE {
  serialNumber            [0]  CertificateSerialNumber OPTIONAL,
  issuer                  [1]  Name OPTIONAL,
  subjectKeyIdentifier    [2]  SubjectKeyIdentifier OPTIONAL,
  authorityKeyIdentifier  [3]  AuthorityKeyIdentifier OPTIONAL,
  certificateValid        [4]  Time OPTIONAL,
  privateKeyValid         [5]  GeneralizedTime OPTIONAL,
  subjectPublicKeyAlgID   [6]  OBJECT IDENTIFIER OPTIONAL,
  keyUsage                [7]  KeyUsage OPTIONAL,
  subjectAltName          [8]  AltNameType OPTIONAL,
  policy                  [9]  CertPolicySet OPTIONAL,
  pathToName              [10] Name OPTIONAL,
  subject                 [11] Name OPTIONAL,
  nameConstraints         [12] NameConstraintsSyntax OPTIONAL,
  ... }

AltNameType ::= CHOICE {
  builtinNameForm  ENUMERATED {
    rfc822Name                (1),
    dNSName                   (2),
    x400Address               (3),
    directoryName             (4),
    ediPartyName              (5),
    uniformResourceIdentifier (6),
    iPAddress                 (7),
    registeredId              (8),
    ...},
  otherNameForm    OBJECT IDENTIFIER,
  ... }

CertPolicySet ::= SEQUENCE SIZE (1..MAX) OF CertPolicyId
```

This matching rule returns TRUE if all of the components that are present in the presented value match the corresponding components of the attribute value, as follows:

`serialNumber` matches if the value of this component in the attribute value equals that in the presented value;

`issuer` matches if the value of this component in the attribute value equals that in the presented value;

`subjectKeyIdentifier` matches if the value of this component in the stored attribute value equals that in the presented value; there is no match if the stored attribute value contains no subject key identifier extension;

`authorityKeyIdentifier` matches if the value of this component in the stored attribute value equals that in the presented value; there is no match if the stored attribute value contains no authority key identifier extension or if not all components in the presented value are present in the stored attribute value;

`certificateValid` matches if the presented value falls within the validity period of the stored attribute value;

`privateKeyValid` matches if the presented value falls within the period indicated by the private key usage period extension of the stored attribute value, or if there is no private key usage period extension in the stored attribute value;

`subjectPublicKeyAlgID` matches if it is equal to the **`algorithm`** component of the **`algorithmIdentifier`** of the **`subjectPublicKeyInformation`** component of the stored attribute value;

`keyUsage` matches if all of the bits set in the presented value are also set in the key usage extension in the stored attribute value, or if there is no key usage extension in the stored attribute value;

`subjectAltName` matches if the stored attribute value contains the subject alternative name extension with an **`AltNames`** component of the same name type as indicated in the presented value;

`policy` matches if at least one member of the **`CertPolicySet`** presented appears in the certificate policies extension in the stored attribute value or if either the presented or stored public-key certificate contains the special value **`anyPolicy`** in the **`policy`** component. There is no match if there is no certificate policies extension in the stored attribute value;

`pathToName` matches unless the public-key certificate has a name constraints extension which inhibits the construction of a certification path to the presented name value;

`subject` matches if the value of this component in the attribute value equals that in the presented value;

nameConstraints matches if the subject names in the stored attribute value are within the name space given by the value of the permitted-subtrees component of the presented value and are not within the name space given by the value of the excluded-subtrees component of the presented value.

13.3.3 Certificate pair exact match

The certificate pair exact match rule compares for equality a presented value with an attribute value of type **CertificatePair**. It uniquely selects a single cross-certificate pair.

```
certificatePairExactMatch MATCHING-RULE ::= {
  SYNTAX  CertificatePairExactAssertion
  ID      id-mr-certificatePairExactMatch }

CertificatePairExactAssertion ::= SEQUENCE {
  issuedToThisCAAssertion  [0]  CertificateExactAssertion OPTIONAL,
  issuedByThisCAAssertion  [1]  CertificateExactAssertion OPTIONAL,
  ... }
  (WITH COMPONENTS { ..., issuedToThisCAAssertion  PRESENT } |
   WITH COMPONENTS { ..., issuedByThisCAAssertion  PRESENT } )
```

This matching rule returns TRUE if the components that are present in the **issuedToThisCAAssertion** and **issuedByThisCAAssertion** components of the presented value match the corresponding components of the **issuedToThisCA** and **issuedByThisCA** components, respectively, in the stored attribute value.

13.3.4 Certificate pair match

The certificate pair match rule compares a presented value with an attribute value of type **CertificatePair**. It selects one or more cross-certificate pairs on the basis of various characteristics of either the **issuedToThisCA** or **issuedByThisCA** certificate of the pair.

```
certificatePairMatch MATCHING-RULE ::= {
  SYNTAX  CertificatePairAssertion
  ID      id-mr-certificatePairMatch }

CertificatePairAssertion ::= SEQUENCE {
  issuedToThisCAAssertion  [0]  CertificateAssertion OPTIONAL,
  issuedByThisCAAssertion  [1]  CertificateAssertion OPTIONAL,
  ... }
  (WITH COMPONENTS {..., issuedToThisCAAssertion  PRESENT } |
   WITH COMPONENTS {..., issuedByThisCAAssertion  PRESENT } )
```

This matching rule returns TRUE if all of the components that are present in the **issuedToThisCAAssertion** and **issuedByThisCAAssertion** components of the presented value match the corresponding components of the **issuedToThisCA** and **issuedByThisCA** components, respectively, in the stored attribute value.

13.3.5 Certificate list exact match

The certificate list exact match rule compares for equality a presented value with an attribute value of type **CertificateList**. It uniquely selects a single CRL.

```
certificateListExactMatch MATCHING-RULE ::= {
  SYNTAX  CertificateListExactAssertion
  ID      id-mr-certificateListExactMatch }

CertificateListExactAssertion ::= SEQUENCE {
  issuer             Name,
  thisUpdate         Time,
  distributionPoint  DistributionPointName OPTIONAL }
```

The **distributionPoint** component, when present, matches if the stored attribute value contains an issuing distribution point extension and the value of this component in the presented value equals the corresponding value, in at least one name form, in that extension.

The rule returns TRUE if the components in the stored attribute value match those in the presented value.

13.3.6 Certificate list match

The certificate list match rule compares a presented value with an attribute value of type **CertificateList**. It selects one or more CRLs based on various characteristics.

```
certificateListMatch MATCHING-RULE ::= {
  SYNTAX  CertificateListAssertion
  ID      id-mr-certificateListMatch }

CertificateListAssertion ::= SEQUENCE {
  issuer                          Name                    OPTIONAL,
  minCRLNumber              [0]   CRLNumber               OPTIONAL,
  maxCRLNumber              [1]   CRLNumber               OPTIONAL,
  reasonFlags                     ReasonFlags             OPTIONAL,
  dateAndTime                     Time                    OPTIONAL,
  distributionPoint         [2]   DistributionPointName   OPTIONAL,
  authorityKeyIdentifier    [3]   AuthorityKeyIdentifier  OPTIONAL,
  ... }
```

The matching rule returns TRUE if all of the components that are present in the presented value match the corresponding components of the stored attribute value, as follows:

`issuer` matches if the value of this component in the attribute value equals that in the presented value;

`minCRLNumber` matches if its value is less than or equal to the value in the CRL number extension of the stored attribute value; there is no match if the stored attribute value contains no CRL number extension;

`maxCRLNumber` matches if its value is greater than or equal to the value in the CRL number extension of the stored attribute value; there is no match if the stored attribute value contains no CRL number extension;

`reasonFlags` matches if any of the bits that are set in the presented value are also set in the **`onlySomeReasons`** components of the issuing distribution point extension of the stored attribute value; there is also a match if the stored attribute value contains no **`reasonFlags`** in the issuing distribution point extension, or if the stored attribute value contains no issuing distribution point extension;

NOTE – Even though a CRL matches on a particular value of **`reasonFlags`**, the CRL may not contain any revocation notices with that reason code.

`dateAndTime` matches if the value is equal to or later than the value in the **`thisUpdate`** component of the stored attribute value and is earlier than the value in the **`nextUpdate`** component of the stored attribute value; there is no match if the stored attribute value contains no **`nextUpdate`** component;

`distributionPoint` matches if the stored attribute value contains an issuing distribution point extension and the value of this component in the presented value equals the corresponding value, in at least one name form, in that extension;

`authorityKeyIdentifier` matches if the value of this component in the stored attribute value equals that in the presented value; there is no match if the stored attribute value contains no authority key identifier extension or if not all components in the presented value are present in the stored attribute value.

13.3.7 Algorithm identifier match

The algorithm identifier match rule compares for equality a presented value with an attribute value of type **`SupportedAlgorithms`**.

```
algorithmIdentifierMatch MATCHING-RULE ::= {
  SYNTAX  AlgorithmIdentifier {{SupportedAlgorithms}}
  ID      id-mr-algorithmIdentifierMatch }
```

The rule returns TRUE if the presented value is equal to the **`algorithmIdentifier`** component of the stored attribute value.

13.3.8 Policy match

The policy match rule compares for equality a presented value with an attribute value of type **`CertificatePolicy`** or an attribute value of type **`privPolicy`**.

```
policyMatch MATCHING-RULE ::= {
  SYNTAX  PolicyID
  ID      id-mr-policyMatch }
```

The rule returns TRUE if the presented value is equal to the **`policyIdentifier`** component of the stored attribute value.

13.3.9 PKI path match

The **`pkiPathMatch`** match rule compares for equality a presented value with an attribute value of type **`pkiPath`**. A relying party may use this matching rule to select a path beginning with a public-key certificate issued by a CA which it trusts and ending with a public-key certificate issued to the specified subject.

```
pkiPathMatch MATCHING-RULE ::= {
  SYNTAX  PkiPathMatchSyntax
  ID      id-mr-pkiPathMatch }

PkiPathMatchSyntax ::= SEQUENCE {
  firstIssuer  Name,
  lastSubject  Name,
  ... }
```

This matching rule returns TRUE if the presented value in the **firstIssuer** component matches the corresponding elements of the **issuer** component of the first public-key certificate in the **SEQUENCE** in the stored value and the presented value in the **lastSubject** component matches the corresponding elements of the **subject** component of the last public-key certificate in the **SEQUENCE** in the stored value. This matching rule returns FALSE if either match fails.

13.3.10 Enhanced certificate match

The enhanced certificate match rule compares a presented value with an attribute value of type **Certificate**. It selects one or more public-key certificates based on various characteristics.

```
enhancedCertificateMatch MATCHING-RULE ::= {
  SYNTAX  EnhancedCertificateAssertion
  ID      id-mr-enhancedCertificateMatch }

EnhancedCertificateAssertion ::= SEQUENCE {
  serialNumber             [0]  CertificateSerialNumber OPTIONAL,
  issuer                   [1]  Name OPTIONAL,
  subjectKeyIdentifier     [2]  SubjectKeyIdentifier OPTIONAL,
  authorityKeyIdentifier   [3]  AuthorityKeyIdentifier OPTIONAL,
  certificateValid         [4]  Time OPTIONAL,
  privateKeyValid          [5]  GeneralizedTime OPTIONAL,
  subjectPublicKeyAlgID    [6]  OBJECT IDENTIFIER OPTIONAL,
  keyUsage                 [7]  KeyUsage OPTIONAL,
  subjectAltName           [8]  AltName OPTIONAL,
  policy                   [9]  CertPolicySet OPTIONAL,
  pathToName               [10] GeneralNames OPTIONAL,
  subject                  [11] Name OPTIONAL,
  nameConstraints          [12] NameConstraintsSyntax OPTIONAL,
  ... }
  (ALL EXCEPT ({ -- none; at least one component shall be present --}))

AltName ::= SEQUENCE {
  altnameType   AltNameType,
  altNameValue  GeneralName OPTIONAL }
```

The directory search operation allows for multiple values of **EnhancedCertificateAssertion** to be combined in filter specifications, including and/or logic. This matching rule returns TRUE if all of the components that are present in the presented value match the corresponding components of the attribute value, as follows:

Matching for **serialNumber**; **issuer**; **subjectKeyIdentifier**; **authorityKeyIdentifier**; **certificateValid**, **privateKeyValid**, **policy**, **subject**, and **nameConstraints** components is as defined for the same components in the **certificateMatch** matching rule.

subjectAltName component contains an **altNameType** and optional **altNameValue** components. If **altNameValue** is present, the value shall be of the same name form as indicated in **altNameType**.

subjectAltName matches if at least one of the following conditions is true:

– The presented value contains only the **altNameType** component and the stored attribute value contains the subject alternative name extension with an **AltNames** component of the same type as indicated in the presented value.

– The presented value contains both the **altNameType** and **altNameValue** components and the stored attribute value contains the subject alternative name extension with an **AltNames** component of the same type and value indicated in the presented value.

subjectAltName match fails if at least one of the following conditions is true:

– The stored attribute value does not contain the subject alternative name extension.

– The stored attribute value contains the subject alternative name extension but the **AltNames** component does not include the same type as identified in the presented value.

– The presented value contains both the **`altNameType`** and **`altNameValue`** components and the stored attribute value contains the subject alternative name extension with an **`AltNames`** component of the same type indicated in the presented value, but the stored value does not contain the same value of that type as in the presented value.

`subjectAltName` match is undefined if the presented value contains both the **`altNameType`** and **`altNameValue`** components and the stored attribute value contains the subject alternative name extension with an **`AltNames`** component of the same type indicated in the presented value, but the type is one for which the directory is unable to compare values for the purposes of determining a match. This may be because the name form is not appropriate for matching or because the directory is unable to perform the required comparisons.

`pathToName` matches unless the public-key certificate has a name constraints extension which inhibits the construction of a certification path to any of the presented name values. For example, if attempting to retrieve public-key certificates that form a path to an end-entity public-key certificate which has a subject value of "dc=com; dc=corporate; cn=john.smith", it may be useful to include an assertion in the search operation containing this distinguished name in the **`pathToName`** component. A stored public-key certificate that contained a name constraints extension that excluded the complete subtree below base "dc=com; dc=company A" would fail in certification path validation to that end-entity public-key certificate and would therefore not be a matched value for this sample assertion.

13.4 PKI directory syntax definitions

13.4.1 X.509 Certificate syntax

```
x509Certificate SYNTAX-NAME ::= {
  DESC                "X.509 Certificate"
  DIRECTORY SYNTAX    Certificate
  ID                  id-lsx-x509Certificate }
```

A value which has LDAP **`x509Certificate`** syntax is the specification of a public-key certificate expressed in a binary encoding as specified in IETF RFC 4523.

13.4.2 X.509 Certificate List syntax

```
x509CertificateList SYNTAX-NAME ::= {
  DESC                "X.509 Certificate List"
  DIRECTORY SYNTAX    CertificateList
  ID                  id-lsx-x509CertificateList }
```

A value which has LDAP **`x509CertificateList`** syntax is the specification of a public-key certificate list expressed in a binary encoding as specified in IETF RFC 4523.

13.4.3 X.509 Certificate Pair syntax

```
x509CertificatePair SYNTAX-NAME ::= {
  DESC                "X.509 Certificate Pair"
  DIRECTORY SYNTAX    CertificatePair
  ID                  id-lsx-x509CertificatePair }
```

A value which has LDAP **`x509CertificatePair`** syntax is the specification of a public-key certificate pair expressed in a binary encoding as specified in IETF RFC 4523.

13.4.4 X.509 Supported Algorithm

```
x509SupportedAlgorithm SYNTAX-NAME ::= {
  DESC                "X.509 Supported Algorithm"
  DIRECTORY SYNTAX    SupportedAlgorithm
  ID                  id-lsx-x509SupportedAlgorithm }
```

A value which has LDAP **`x509SupportedAlgorithm`** syntax is the specification of supported algorithms expressed in a binary encoding as specified in IETF RFC 4523.

13.4.5 X.509 Certificate Exact Assertion

```
x509CertificateExactAssertion SYNTAX-NAME ::= {
  DESC                "X.509 Certificate Exact Assertion"
  DIRECTORY SYNTAX    CertificateExactAssertion
  ID                  id-ldx-x509CertificateExactAssertion }
```

A value which has LDAP **`x509CertificateExactAssertion`** syntax is the specification of public-key exact assertion expressed in a Generic String Encoding Rules encoding as specified in IETF RFC 4523.

13.4.6 X.509 Certificate Assertion

```
x509CertificateAssertion SYNTAX-NAME ::= {
  DESC                "X.509 Certificate Assertion"
  DIRECTORY SYNTAX    CertificateAssertion
  ID                  id-ldx-x509CertificateAssertion }
```

A value which has LDAP **`x509CertificateAssertion`** syntax is the specification of public-key assertion expressed in a Generic String Encoding Rules encoding as specified in IETF RFC 4523.

13.4.7 X.509 Certificate Pair Exact Assertion

```
x509CertificatePairExactAssertion SYNTAX-NAME ::= {
  DESC                "X.509 Certificate Pair Exact Assertion"
  DIRECTORY SYNTAX    CertificatePairExactAssertion
  ID                  id-ldx-x509CertificatePairExactAssertion }
```

A value which has LDAP **`x509CertificatePairExactAssertion`** syntax is the specification of public-key certificate pair exact assertion expressed in a Generic String Encoding Rules encoding as specified in IETF RFC 4523.

13.4.8 X.509 Certificate Pair Assertion

```
x509CertificatePairAssertion SYNTAX-NAME ::= {
  DESC                "X.509 Certificate Pair Assertion"
  DIRECTORY SYNTAX    CertificatePairAssertion
  ID                  id-ldx-x509CertificatePairAssertion }
```

A value which has LDAP **`x509CertificatePairAssertion`** syntax is the specification of a public-key certificate pair assertion expressed in a Generic String Encoding Rules encoding as specified in IETF RFC 4523.

13.4.9 X.509 Certificate List Exact Assertion syntax

```
x509CertificateListExactAssertion SYNTAX-NAME ::= {
  DESC                "X.509 Certificate List Exact Assertion"
  DIRECTORY SYNTAX    CertificateListExactAssertion
  ID                  id-ldx-x509CertListExactAssertion }
```

A value which has LDAP **`x509CertificateListExactAssertion`** syntax is the specification of a public-key certificate list exact assertion expressed in a Generic String Encoding Rules encoding as specified in IETF RFC 4523.

13.4.10 X.509 Certificate List Assertion syntax

```
x509CertificateListAssertion SYNTAX-NAME ::= {
  DESC                "X.509 Certificate List Assertion"
  DIRECTORY SYNTAX    CertificateListAssertion
  ID                  id-ldx-x509CertificateListAssertion }
```

A value which has LDAP **`x509CertificateListAssertion`** syntax is the specification of a public-key certificate list assertion expressed in a Generic String Encoding Rules encoding as specified in IETF RFC 4523.

13.4.11 X.509 Algorithm Identifier syntax

```
x509AlgorithmIdentifier SYNTAX-NAME ::= {
  DESC                "X.509 Algorithm Identifier"
  DIRECTORY SYNTAX    AlgorithmIdentifier{{SupportedAlgorithms}}
  ID                  id-ldx-x509AlgorithmIdentifier }
```

A value which has LDAP **`x509AlgorithmIdentifie`** syntax is the specification of an algorithm identifier expressed in a binary encoding as specified in IETF RFC 4523.

SECTION 3 – ATTRIBUTE CERTIFICATE FRAMEWORK

The attribute certificate framework defined here provides a foundation upon which privilege management infrastructures (PMI) can be built. These infrastructures can support applications such as access control.

The binding of a privilege to an entity is provided by an authority through a digitally signed data structure called an attribute certificate or through a public-key certificate containing the **`subjectDirectoryAttributes`** extension defined explicitly for this purpose. The format of attribute certificates is defined here, including an extensibility mechanism and a set of specific attribute certificate extensions. Some of these extensions defined in Section 2 are also relevant for attribute certificates. Revocation of attribute certificates may or may not be needed. For example, in some environments, the attribute certificate validity periods may be very short (e.g., minutes), negating the need for a revocation scheme. If, for any reason, an authority revokes a previously issued attribute certificate, users relying on the content of an attribute certificate (privilege verifier) need to be able to learn that revocation has occurred so they do not use an untrustworthy attribute certificate. Revocation lists are one scheme that can be used to notify privilege verifiers of revocations. The format of revocation lists is defined in Section 2 of this Specification, including an extensibility mechanism and a set of revocation list extensions. Some of these extensions are also relevant for attribute certificate revocation lists. Additional extensions are defined here. In both the attribute certificate and revocation list case, other bodies may also define additional extensions that are useful to their specific environments.

An entity relying on an attribute certificate to verify some asserted privilege is called a privilege verifier. A privilege verifier needs to validate an attribute certificate prior to using that attribute certificate for an application. Procedures for performing that validation are also defined here, including verifying the integrity of the attribute certificate itself, its revocation status, and its validity with respect to the intended use.

This framework includes a number of optional elements that are appropriate only in some environments. Although the models are defined as complete, this framework can be used in environments where not all components of the defined models are used. For example, there are environments where the revocation of attribute certificates is not required. Privilege delegation and the use of roles are also aspects of this framework that are not universally applicable. However, these are included in this Specification so that those environments that do have requirements for them can also be supported.

14 Attribute certificates

Public-key certificates are principally intended to provide an identity service upon which other security services, such as data integrity, entity authentication, confidentiality and authorization, may be built. There are two distinct mechanisms provided in this Specification for binding a privilege attribute to a holder.

Public-key certificates, used in combination with the entity authentication service, can provide an authorization service directly, if privileges are associated with the subject. Public-key certificates may contain a **`subjectDirectoryAttributes`** extension that contains privileges associated with the subject of the public-key certificate (see clause 9.3.2.3). This mechanism is appropriate in situations where the CA issuing the public-key certificate has an associated authority for delegating the privilege (i.e., an AA) and the validity period of the privilege corresponds to the validity period of the public-key certificate. If any of the extensions defined in clause 17 are included in a public-key certificate, those extensions apply equally to all privileges assigned in the **`subjectDirectoryAttributes`** extension of that public-key certificate.

In the more general case, entity privileges will have lifetimes that do not match the validity period for a public-key certificate. Privileges will often have a much shorter lifetime. The authority for the assignment of privilege will frequently be one other than the authority issuing that same entity a public-key certificate and different privileges may be assigned by different attribute authorities (AA). Privileges may also be assigned based on a temporal context and the 'turn on/turn off' aspect of privileges may well be asynchronous with the lifetime of the public-key certificate and/or asynchronous with entity privileges issued from a different AA. The use of attribute certificates issued by an AA provides a flexible privilege management infrastructure (PMI) which can be established and managed independently from a PKI. At the same time, there is a relationship between the two whereby the PKI is used to authenticate identities of issuers and holders in attribute certificates.

14.1 Attribute certificate structure

An attribute certificate is a separate structure from a subject's public-key certificate. A subject may have multiple attribute certificates associated with each of its public-key certificates. There is no requirement that the same authority create both the public-key certificate and attribute certificate(s) for an entity; in fact, a separation of duties will frequently dictate otherwise. In environments where different authorities have responsibility for issuing public key and attribute certificates, the public-key certificate(s) issued by a CA and the attribute certificate(s) issued by an attribute authority (AA) would be signed using different private signing keys. In environments where a single entity is both the CA, issuing public-key certificates, and the AA, issuing attribute certificates, it is strongly recommended that a different key be used to sign attribute certificates than the key used to sign public-key certificates. Exchanges between the issuing authority and the entity receiving an attribute certificate are outside the scope of this Specification.

The attribute certificate syntax is defined as follows.

```
AttributeCertificate ::= SIGNED{TBSAttributeCertificate}

TBSAttributeCertificate ::= SEQUENCE {
  version                 AttCertVersion, -- version is v2
  holder                  Holder,
  issuer                  AttCertIssuer,
  signature               AlgorithmIdentifier{{SupportedAlgorithms}},
  serialNumber            CertificateSerialNumber,
  attrCertValidityPeriod  AttCertValidityPeriod,
  attributes              SEQUENCE OF Attribute{{SupportedAttributes}},
  issuerUniqueID          UniqueIdentifier OPTIONAL,
  ...,
  ...,
  extensions              OPTIONAL }

AttCertVersion ::= INTEGER {v2(1)}

Holder ::= SEQUENCE {
  baseCertificateID   [0]  IssuerSerial OPTIONAL,
  entityName          [1]  GeneralNames OPTIONAL,
  objectDigestInfo    [2]  ObjectDigestInfo OPTIONAL }
  (WITH COMPONENTS {..., baseCertificateID  PRESENT } |
   WITH COMPONENTS {..., entityName  PRESENT } |
   WITH COMPONENTS {..., objectDigestInfo  PRESENT } )

IssuerSerial ::= SEQUENCE {
  issuer     GeneralNames,
  serial     CertificateSerialNumber,
  issuerUID  UniqueIdentifier OPTIONAL,
  ... }

ObjectDigestInfo ::= SEQUENCE {
  digestedObjectType   ENUMERATED {
    publicKey         (0),
    publicKeyCert     (1),
    otherObjectTypes  (2)},
  otherObjectTypeID   OBJECT IDENTIFIER OPTIONAL,
  digestAlgorithm     AlgorithmIdentifier{{SupportedAlgorithms}},
  objectDigest        BIT STRING,
  ... }

AttCertIssuer ::= [0]  SEQUENCE {
  issuerName               GeneralNames OPTIONAL,
  baseCertificateID   [0]  IssuerSerial OPTIONAL,
  objectDigestInfo    [1]  ObjectDigestInfo OPTIONAL,
  ... }
  (WITH COMPONENTS {..., issuerName  PRESENT } |
   WITH COMPONENTS {..., baseCertificateID  PRESENT } |
   WITH COMPONENTS {..., objectDigestInfo  PRESENT } )

AttCertValidityPeriod ::= SEQUENCE {
  notBeforeTime  GeneralizedTime,
  notAfterTime   GeneralizedTime,
  ... }
```

The **TBSAttributeCertificate** data type is the unsigned attribute certificate and is referred to as a to-be-signed attribute certificate. It shall be encoded using the DER.

The **version** component shall specify the version of the attribute certificate. For attribute certificates issued in accordance with the syntax in this Specification, **version** shall be **v2**.

The **holder** component shall convey the identity of the holder of the attribute certificate by the following components:

a) The **baseCertificateID** component, if present, shall identify a particular public-key certificate that is to be used to authenticate the identity of this holder when asserting privileges with this attribute certificate.

b) The **`entityName`** component, if present, shall hold one or more names for the holder. If **`entityName`** is the only component present in **`holder`**, any public-key certificate that has one of these names as its subject can be used to authenticate the identity of this holder when asserting privileges with this attribute certificate. If the **`baseCertificateID`** and **`entityName`** components are both present, only the public-key certificate specified by **`baseCertificateID`** may be used. In this case, the **`entityName`** component is included only as a tool to help the privilege verifier locate the identified public-key certificate.

NOTE 1 – There is a risk with the sole use of **`GeneralNames`** to identify the holder in that this points only to a name for the holder. This is generally insufficient to enable the authentication of a holder's identity for the purposes of issuing privileges to that holder. Use of the issuer name and serial number of a specific public-key certificate, however, enables the issuer of attribute certificates to rely on the authentication process performed by the CA when issuing that particular public-key certificate. Also, some of the options in **`GeneralNames`** (e.g., **`IPAddress`**) are inappropriate for use in naming an attribute certificate holder, especially when the holder is a role and not an individual entity. Another problem with **`GeneralNames`** alone as an identifier for a holder is that many name forms within that construct do not have strict registration authorities or processes for the assignment of names.

c) The **`objectDigestInfo`** component, if present, is used directly to authenticate the identity of a holder, including an executable holder (e.g., an applet). The holder is authenticated by comparing a digest of the corresponding information, created by the privilege verifier with the same algorithm identified in **`objectDigestInfo`** with the content of **`objectDigest`**. If the two are identical, the holder is authenticated for the purposes of asserting privileges with this attribute certificate.

– **`publicKey`** shall be indicated when a hash of an entity's public key is included. Hashing a public key may not uniquely identify one public-key certificate (i.e., the identical key value may appear in multiple public-key certificates). In order to link an attribute certificate to a public key, the hash is calculated over the representation of that public key which would be present in a public-key certificate. Specifically, the input for the hash algorithm shall be the DER encoding of a **`SubjectPublicKeyInfo`** representation of the key. Note that this includes the **`AlgorithmIdentifier,`** as well as the **`BIT STRING`**. Also, note that if the public key value used as input to the hash function has been extracted from a public-key certificate, then it is possible (e.g., if parameters for the digital signature algorithm were inherited) that this may not be sufficient input for the HASH. The correct input for hashing in this context will include the value of the inherited parameters and thus may differ from the **`SubjectPublicKeyInfo`** present in the public-key certificate.

– **`publicKeyCert`** shall be indicated when a public-key certificate is hashed; the hash is over the entire DER encoding of the public-key certificate, including the signature bits.

– **`otherObjectTypes`** shall be indicated when objects other than public-keys or public-key certificates are hashed (e.g., software objects). The identity of the type of object may optionally be supplied. The portion of the object to be hashed can be determined either by the explicitly stated identifier of the type or, if the identifier is not supplied, by the context in which the object is used.

The **`issuer`** component shall convey the identity of the AA that issued the attribute certificate.

a) The **`issuerName`** component, if present, shall identify one or more names for the issuer.

b) The **`baseCertificateID`** component, if present, shall identify the issuer by reference to a specific public-key certificate for which this issuer is the subject.

c) The **`objectDigestInfo`** component, if present, shall identify the issuer by providing a hash of identifying information for the issuer.

The **`signature`** component shall identify the cryptographic algorithm used to digitally sign the attribute certificate.

NOTE 2 – By including this component, the signature algorithm is protected by the signature.

The **`serialNumber`** component shall be a serial number that uniquely identifies the attribute certificate within the scope of its issuer.

The **`attrCertValidityPeriod`** component shall convey the time period during which the attribute certificate is considered valid, expressed in **`GeneralizedTime`** format.

The **`attributes`** component shall contain the attributes associated with the holder that are being certified (e.g., the privileges).

NOTE 3 – In the case of attribute descriptor attribute certificates, this sequence of attributes can be empty.

The **`issuerUniqueID`** component may be used to identify the issuer of the attribute certificate in instances where the issuer component is not sufficient.

NOTE 4 – The use of the **`issuerUniqueID`** component is deprecated. This component was added because at one time there was some fear of the reuse of distinguished names.

The **extensions** component, when present, shall hold one or more extensions as defined in clause 17.

If unknown elements appear within the extension, and the extension is not flagged as critical, those unknown elements shall be ignored according to the rules of extensibility documented in clause 12.2.2 of Rec. ITU-T X.519 | ISO/IEC 9594-5.

The framework for attribute certificates described in this section is primarily focused on the model in which privilege is placed within attribute certificates. Some of the attribute certificate extensions defined in in clause 17 can also be placed in a public-key certificate when including the **subjectDirectoryAttributes** extension.

14.2 Delegation paths

Just as with public-key certificates, there may be a requirement to convey a delegation path (e.g., within an application protocol to assert privileges). The following ASN.1 data type can be used to represent a delegation path:

```
AttributeCertificationPath ::= SEQUENCE {
  attributeCertificate   AttributeCertificate,
  acPath                 SEQUENCE OF ACPathData OPTIONAL,
  ... }

ACPathData ::= SEQUENCE {
  certificate            [0]  Certificate OPTIONAL,
  attributeCertificate   [1]  AttributeCertificate OPTIONAL,
  ... }
```

14.3 Attribute certificate revocation lists

14.3.1 Attribute certificate revocation list principles

The AA that issues attribute certificates also has the responsibility to indicate the validity of the attribute certificates that it issues. Generally, attribute certificates are subject to possible subsequent revocation. This revocation and a notification of the revocation may be done directly by the same AA that issued the attribute certificate, or indirectly by another authority duly authorized by the AA that issued the attribute certificate. An AA that issues attribute certificates is required to state, possibly through a published statement of their practices, through the attribute certificates themselves, or through some other identified means, whether:

– the attribute certificates cannot be revoked;

– the attribute certificates may be revoked by the same issuing AA directly; or

– the issuing AA authorizes a different entity to perform revocation.

AAs that do revoke attribute certificates are required to state, through similar means, what mechanism(s) can be used by privilege verifiers to obtain revocation status information about attribute certificates issued by that AA. This Specification defines an attribute certificate revocation list (ACRL) mechanism but does not preclude the use of alternative mechanisms. Privilege verifiers check revocation status information, as appropriate, for all attribute certificates considered during the delegation path processing procedure described in clause 18 to validate an attribute certificate.

Only an AA that is authorized to issue ACRLs may choose to delegate that authority to another entity. If this delegation is done, it shall be verifiable at the time of attribute certificate/ACRL verification. The **cRLDistributionPoints** extension (see clause 17.2.2.1) may be used for this purpose. The **cRLIssuer** component of this extension is populated with the name(s) of any entities, other than the issuing AA itself, that have been authorized to issue CRLs concerning the revocation status of the attribute certificate in question.

Attribute certificates shall have a lifetime associated with them, at the end of which they expire. In order to provide continuity of service, the AA shall ensure timely availability of replacement attribute certificates to supersede expired/expiring attribute certificates. Revocation notice date is the date/time that a revocation notice for an attribute certificate first appears on an ACRL, regardless of whether it is a base or dACRL. In the ACRL, revocation notice date is the value contained in the **thisUpdate** component. Revocation date is the date/time the AA actually revoked the attribute certificate, which could be different from the first time it appears on an ACRL. In the ACRL, revocation date is the value contained in the **revocationDate** component. Invalidity date is the date/time at which it is known or suspected that the attribute certificate should be considered invalid. This date may be earlier than the revocation date. In the ACRL, invalidity date is the value contained in the **invalidityDate** entry extension.

Two related points are:

– Validity of attribute certificates may be designed so that each becomes valid at the time of expiry of its predecessor, or an overlap may be allowed. The latter prevents the AA from having to install and distribute a large number of attribute certificates that may run out at the same expiration date.

– Expired attribute certificates will normally be removed from a directory. It is a matter for the security policy and responsibility of the AA to keep old attribute certificates for a period if needed for later verification.

Attribute certificates may be revoked prior to their expiration time, e.g., if the entity no longer holds the privilege delegated to it by the AA, or if the AA's public-key certificate is assumed to be compromised.

An AA that issues and subsequently revokes attribute certificates:

a) may be required to maintain an audit record of its revocation events for all attribute certificate types issued by that AA (e.g., end-entity attribute certificates, as well as attribute certificates to other AAs);

b) shall provide revocation status information to privilege verifiers using ACRLs or another mechanism for the publication of revocation status information;

c) if using ACRLs, it shall maintain and publish ACRLs even if the lists of revoked attribute certificates are empty;

d) if using only partitioned ACRLs, it shall issue a full set of partitioned ACRLs covering the complete set of attribute certificates whose revocation status will be reported using the ACRL mechanism. Thus, the complete set of partitioned ACRLs shall be equivalent to a full ACRL for the same set of attribute certificates, if the ACRL issuer was not using partitioned ACRLs.

Privilege verifiers may use a number of mechanisms to locate revocation status information provided by an AA. For example, there may be a pointer in the attribute certificate itself that directs the privilege verifier to a location where revocation information is provided. There may be a pointer in a revocation list that redirects the privilege verifier to a different location. The privilege verifier may locate revocation information in a repository (e.g., a directory) or through other means outside the scope of this Specification (e.g., locally configured).

The maintenance of directory entries affected by the AA's revocation lists is the responsibility of the directory and its users, acting in accordance with the security policy. For example, the user may modify its object entry by replacing the old attribute certificate with a new one.

If revocation lists are published in a directory, they are held within entries as attributes of the following types:

– **attributeCertificateRevocationList**;

– **eeAttrCertificateRevocationList**;

– **attributeAuthorityRevocationList**; and

– **deltaRevocationList**.

14.3.2 Attribute certificate revocation list syntax

The syntax for ACRLs is the same as for CRLs as defined in clause 7.10.2.

The associated text is the same with the following exceptions:

– CA shall be replaced with AA;

– public-key certificate shall be replaced with attribute certificate;

– CRL shall be replaced with ACRL;

– NOTE 3 is not relevant for attribute certificates; and

– extensions for ACRLs are defined in clause 17.

15 Attribute authority, source of authority and certification authority relationship

The attribute authority (AA) and certification authority (CA) are logically completely independent. The creation and maintenance of "identity" can (and often should) be separated from the PMI. Thus the entire PKI, including CAs, may be existing and operational prior to the establishment of the PMI. The CA, although it is the source of authority (SOA) for identity within its domain, is not automatically the SOA for privilege. The CA, therefore, will not necessarily itself be an AA and, by logical implication, will not necessarily be responsible for the decision as to what other entities will be able to function as AAs.

The SOA is the entity that is trusted by a privilege verifier as the entity with ultimate responsibility for the assignment of a set of privileges. A resource may limit the SOA authority by trusting certain SOAs for specific functions (e.g., one for read privileges and a different one for write privileges). An SOA is itself an AA as it issues attribute certificates to other entities in which privileges are assigned to those entities. An SOA is analogous to a trust anchor in the PKI, in that a privilege verifier trusts attribute certificates signed by the SOA. In some environments there is a need for CAs to have tight control over the entities that can act as SOAs. This framework provides a mechanism for supporting that requirement. In other environments, that control is not needed and mechanisms for determining the entities that can act as SOAs in such environments may be outside the scope of this Specification.

This framework is flexible and can satisfy the requirements of many types of environments.

a) In many environments, all privileges will be assigned directly to individual entities by a single AA, namely the SOA.

b) Other environments may require support for the optional roles feature, whereby entities are issued attribute or public-key certificates that assign various roles to them. The privileges associated with the role are implicitly assigned to such entities. The role privileges may themselves be assigned in an attribute certificate issued to the role itself or through other means (e.g., locally configured).

c) In some scenarios it might be required for an AA to issue privileges to a group of entities that share a common property, for example, a set of web servers or a team of people, rather than to a single entity.

d) Another optional feature of this framework is the support of privilege delegation. If delegation is done, the SOA assigns privilege to an entity that is permitted to also act as an AA and further delegate the privilege. Delegation may continue through several intermediary AAs until it is ultimately assigned to an entity acting as an PMI end entity that cannot further delegate that privilege. The intermediary AAs may or may not also be able to act as privilege asserters for the privileges they delegate.

e) When identity and/or privilege information is conveyed within the **subjectDirectoryAttributes** extension of a public-key certificate, the AA is then responsible for those aspects of the CA that relate to assigning identity and/or privilege information. The AA may either be a separate entity or and integrated part of the CA.

f) Some environments, such as virtual organizations, may need to link together their individual PMIs to form a federated PMI. This requirement is known as recognition of authority in this Specification since one PMI (the local PMI) recognizes the authority of the SOA (and optionally the AAs) in the other PMI (the remote PMI) to have some control over the privilege management in the local PMI. Such recognition of authority may or may not be mutual between PMIs.

When attribute certificates point to public-key certificates for their issuers and holders, the PKI is used to authenticate holders (privilege asserters) and verify the digital signatures of the issuers.

Two delegation models are described in this Specification. The first delegation model is one where the privilege delegator is an AA that can issue attribute certificates delegating that privilege to others. The second model allows for an independent delegation service (DS) in which the entity issues attribute certificates on behalf of another AA (that may or may not be able to issue attribute certificates itself). This DS cannot itself act as a claimant for that privilege. The DS model is particularly relevant to environments that wish to maintain some central management over the set of privileges delegated within their domain. For example, a set of one or more DS servers performing delegation, rather than individual privilege holders, allows the total set of privileges delegated within an environment to be determined from a centralized facility and enables policy and management decisions to be modified accordingly. Two distinct deployment models are possible for DS servers. In one model, a privilege is assigned by an SOA to privilege holders and those holders are authorized to delegate that privilege to others. However, rather than issue the attribute certificates that delegate the privilege themselves, the privilege holder requests the DS to delegate that privilege on their behalf. The DS does not itself hold that privilege and therefore cannot act as a claimant for that privilege; however, the DS is authorized by the SOA to issue attribute certificates on behalf of other privilege holders. The second deployment model is similar to the first with the following exception. The DS is actually a holder that is assigned the privilege to be delegated, but the DS is not authorized to act as a claimant for the privilege, only as a delegator. In this case, the **noAssertion** extension must be set in the attribute certificate issued to the DS by the SOA. The DS is termed an indirect issuer.

In both deployment models, the SOA issues attributes/privileges to subordinate AAs. The AAs then request the DS to issue a subset of these privilege attributes to other holders. In the second deployment model, the DS can check that an AA is delegating within the overall scope set by the SOA; in the first deployment model, the DS cannot check and the privilege verifier will have to check that delegation was performed correctly.

Two recognition of authority (RoA) models are described in this Specification, static RoA and dynamic RoA. With static RoA, extra information is added into the local PMI policy that is loaded into the local policy decision point (PDP) prior to them making access control decisions for users who originate from the remote domain. No support for static RoA is provided in this Specification. With dynamic RoA, the local SOA issues new supplementary policy attribute certificates that add additional information to the current policy. Remote SOAs may also be recognized to issue supplementary policy attribute certificates for the local PDPs. In both cases, these new supplementary policy attribute certificates need to be read in by the local PDPs prior to them making access control decisions for requests from a user of the remote domain.

15.1 Privilege in attribute certificates

Entities may acquire privilege in two ways:

- An AA may unilaterally assign privilege to an entity through the creation of an attribute certificate (perhaps totally on its own initiative, or at the request of a third party). This attribute certificate may be stored in a publicly accessible repository and may subsequently be processed by one or more privilege verifiers to make an authorization decision. All of this may occur without the entity's knowledge or explicit action.
- Alternatively, an entity may request a privilege of an AA. Once created, this attribute certificate may be returned (only) to the requesting entity, which explicitly supplies it when requesting access to a protected resource.

Note that in both procedures the AA needs to perform its due diligence to ensure that the entity should really be assigned this privilege. This may involve some out-of-band mechanisms, analogous to the certification of an identity/key-pair binding by a CA.

The attribute certificate based PMI is suitable in environments where any one of the following is true:

- A different entity is responsible for assigning a particular privilege to a holder than for issuing public-key certificates to the same subject;
- there are a number of privilege attributes to be assigned to a holder, from a variety of authorities;
- the lifetime of a privilege differs from that of the holder's public-key certificate validity (generally the lifetime of privileges is much shorter); or
- the privilege is valid only during certain intervals of time which are asynchronous with that user's public-key validity or validity of other privileges.

15.2 Privilege in public-key certificates

In some environments, privilege may be put directly into public-key certificates (thereby reusing much of an already-established infrastructure), rather than issuing attribute certificates. In such cases, the privilege is included in the **`subjectDirectoryAttributes`** extension of the public-key certificate.

This mechanism is suitable in environments where one or more of the following are true:

- the lifetime of the privilege is aligned with that of the public-key included in the certificate;
- delegation of privilege is not permitted; or
- delegation is permitted, but for any one delegation, all privileges in the public-key certificate (in the **`subjectDirectoryAttributes`** extension) have the same delegation parameters and all extensions relevant to delegation apply equally to all privileges in the public-key certificate.

16 PMI models

16.1 General model

The general privilege management model consists of three entities: the object, the privilege asserter and the privilege verifier.

The object may be a resource being protected, for example, in an access control application. The resource being protected is referred to as the object. This type of object has methods which may be invoked (for example, the object may be a firewall which has an "Allow Entry" object method, or the object may be a file in a file system which has Read, Write, and Execute object methods). Another type of object in this model may be an object that was signed in a non-repudiation application.

The privilege asserter is the entity that holds a particular privilege and asserts its privileges for a particular context of use.

The privilege verifier is the entity that makes the determination as to whether or not asserted privileges are sufficient for the given context of use.

The pass/fail determination made by the privilege verifier is dependent upon four things:

- privilege of the privilege asserter;
- privilege policy in place;
- current environment variables, if relevant; and
- sensitivity of the object method, if relevant.

The privilege of a privilege holder reflects the degree of trust placed in that holder, by the certificate issuer, that the privilege holder will adhere to those aspects of policy which are not enforced by technical means. This privilege is encapsulated in the privilege holder's attribute certificate(s) (or **`subjectDirectoryAttributes`** extension of its public-key certificate), which may be presented to the privilege verifier in the invocation request, or may be distributed by other means, such as via a directory. Codifying privilege is done through the use of the **`Attribute`** construct, containing an **`AttributeType`** and a **`SET OF AttributeValue`**. Some attribute types used to specify privilege may have very simple syntax, such as a single **`INTEGER`** or an **`OCTET STRING`**. Others may have more complex syntaxes. This Specification defines one simple privilege attribute type. Other examples are provided in Annex G.

The privilege policy specifies the degree of privilege which is considered sufficient for a given object method's sensitivity or context of use. The privilege policy needs to be protected for integrity and authenticity. A number of possibilities exist for conveying policy. At one extreme is the idea that policy is not really conveyed at all, but is simply defined and only ever kept locally in the privilege verifier's environment. At the other extreme is the idea that some policies are "universal" and should be conveyed to, and known by, every entity in the infrastructure. Between these extremes are many shades of variation. Schema elements for storing privilege policy information in a directory are defined in this Specification.

Privilege policy specifies the threshold for acceptance for a given set of privileges. That is, it defines precisely when a privilege verifier should conclude that a presented set of privileges is "sufficient" in order that it may grant access (to the requested object, resource, application, etc.) to the privilege asserter.

Syntax for the definition of privilege policy is not standardized in this Specification. Annex G contains a couple of examples of syntaxes that could be used for this purpose. However, these are examples only. Any syntax may be used for this purpose, including clear text. Regardless of the syntax used to define the privilege policy, each instance of privilege policy shall be uniquely identified. Object identifiers are used for this purpose.

```
PrivilegePolicy ::= OBJECT IDENTIFIER
```

The environment variables, if relevant, capture those aspects of policy required for the pass/fail determination (e.g., time of day or current account balance) which are available through local means to the privilege verifier. Representation of environment variables is entirely a local matter.

The object method sensitivity, if relevant, may reflect attributes of the document or request to be processed, such as the monetary value of a funds transfer that it purports to authorize, or the confidentiality of a document's content. The object method's sensitivity may be explicitly encoded in an associated security label or in an attribute certificate held by the object method, or it may be implicitly encapsulated in the structure and contents of the associated data object. It may be encoded in one of a number of different ways. For instance, it may be encoded outside the scope of the PMI in the ITU-T X.411 label associated with a document, in the components of an EDIFACT interchange, or hard-coded in the privilege verifier's application. Alternatively, it may be done within the PMI, in an attribute certificate associated with the object method. For some contexts of use, no object method sensitivity is used.

There is not necessarily any binding relationship between a privilege verifier and any particular AA. Just as privilege holders may have attribute certificates issued to them by many different AAs, privilege verifiers may accept attribute certificates issued by numerous AAs, which need not be hierarchically related to one another, to grant access to a particular resource.

The attribute certificate framework can be used to manage privileges of various types and for a number of purposes. The terms used in this Specification, such as privilege asserter, privilege verifier, etc., are independent of the particular application or use.

16.1.1 PMI in access control context

There is a standard framework for access control (Rec. ITU-T X.812 | ISO/IEC 10181-3) that defines a corresponding set of terms that are specific to the access control application. A mapping of the generic terms used in this Specification to those in the access control framework is provided here, to clarify the relationship between this model and this Specification.

Service request in this Specification corresponds to the 'access request' defined in the access control framework.

Privilege asserter in this Specification would be acting in the role of an 'initiator' in the access control framework.

Privilege verifier in this Specification would be acting in the role of an 'access control decision function (ADF)' in the access control framework.

Object method for which privilege is being asserted in this Specification would correspond to the 'target' defined in the access control framework.

Environmental variables in this Specification would correspond to the 'contextual information' in the access control framework.

Privilege policy discussed in this Specification could include 'access control policy', and 'access control policy rules' as defined in the access control framework.

This model allows a PMI to be overlaid fairly seamlessly on an existing network of resources to be protected. In particular, having the privilege verifier act as a gateway to a sensitive object method, granting or denying requests for invocation of that object method, enables the object to be protected with little or no impact to the object itself. The privilege verifier screens all requests and only those that are properly authorized are passed on to the appropriate object methods.

16.1.2 PMI in a non-repudiation context

There is a standard framework for non-repudiation (Rec. ITU-T X.813 | ISO/IEC 10181-4) which defines a corresponding set of terms that are specific to non-repudiation. A mapping of the generic terms used in this Specification to those in the non-repudiation framework is provided here, to clarify the relationship between this model and that Specification.

Privilege asserter in this Specification would be acting in the role of an 'evidence subject' or an 'originator' in the non-repudiation framework.

Privilege verifier in this Specification would be acting in the role of an 'evidence user' or a 'recipient' in the non-repudiation framework.

Object method for which privilege is being asserted in this Specification would correspond to the 'target' defined in the non-repudiation framework.

Environmental variables in this Specification would correspond to the date and time the evidence was generated or verified' in the non-repudiation framework.

Privilege policy discussed in this Specification could include 'non-repudiation security policy' in the non-repudiation framework.

16.2 Control model

The control model illustrates how control is exerted over access to the sensitive object method. There are five components of the model: the privilege asserter, the privilege verifier, the object method, the privilege policy, and environmental variables (see Figure 6). The privilege asserter has privilege; the object method has sensitivity. The techniques described here enable the privilege verifier to control access to the object method by the privilege asserter, in accordance with the privilege policy. Both the privilege and the sensitivity may be multi-valued parameters.

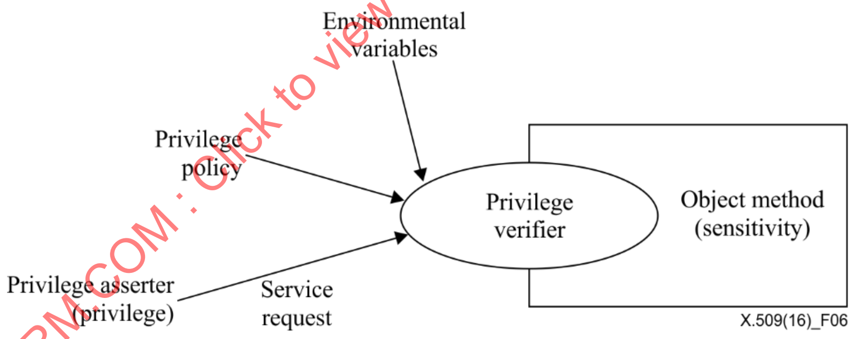

Figure 6 – Control model

The privilege asserter may be an entity identified by a public-key certificate, or an executable object identified by the digest of its disk image, etc.

16.3 Delegation model

In some environments there may be a need to delegate privilege; however, this is an optional aspect of the framework and is not required in all environments. There are four components of the delegation model: the privilege verifier, the SOA, other AAs and the privilege asserter (see Figure 7).

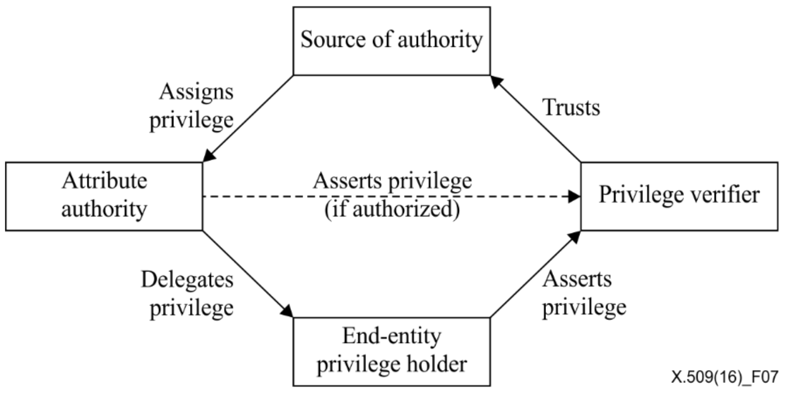

Figure 7 – Delegation model

As with environments where delegation is not used, the SOA is the initial issuer of attribute certificates that assign privilege to privilege holders. However, in this case the SOA authorizes the privilege holder to act as AA and further delegate that privilege to other entities through the issuance of attribute certificates that contain the same privilege (or a subset thereof). The SOA may impose constraints on the delegation that can be done (e.g., limit the path length, limit the name space within delegation can be done). Each of these intermediary AAs may, in attribute certificates that it issues to further privilege holders, authorize further delegation to be done by those holders also acting as AAs. A universal restriction on delegation is that no AA can delegate more privilege than it holds. A delegator may also further restrict the ability of downstream AAs.

When delegation is used, the privilege verifier trusts the SOA to delegate some or all of those privileges to holders, some of which may further delegate some or all of those privileges to other holders.

The privilege verifier trusts the SOA as the authority for a given set of privileges for the resource. If the privilege asserter's certificate is not issued by that SOA, then the privilege verifier shall locate a delegation path of certificates from that of the privilege asserter to one issued by the SOA. The validation of that delegation path includes checking that each AA had sufficient privileges and was duly authorized to delegate those privileges.

For the case in which privileges are conveyed by means of attribute certificates, the delegation path is distinct from the certification path used to validate the public-key certificates of the entities involved in the delegation process. However, the quality of authenticity offered by the public-key certificate validation process shall be commensurate with the sensitivity of the object method that is being protected.

A delegation path shall either consist completely of attribute certificates or completely of public-key certificates. A delegator that obtains its privilege in an attribute certificate may only delegate, if authorized, by issuance of subsequent attribute certificates. Similarly, a delegator that obtains its privilege in a public-key certificate, if authorized, may only delegate by issuance of subsequent public-key certificates. Only entities acting as AAs may delegate privilege. Entities acting as PMI end entities cannot.

16.4 Group assignment model

In some scenarios it might be required for an AA to issue privileges to a group of entities that share a common property, for example, a set of web servers or a team of people, rather than to a single entity. This is achieved by assigning a group attribute certificate to the group.

There are two ways of identifying the members of a group who are assigned a group attribute certificate. These methods are called direct group naming and group role naming.

16.4.1 Direct group naming

In direct group naming, the **`holder`** component of the group attribute certificate shall take the **`entityName`** option, and the **`directoryName`** of **`GeneralName`** shall name a subtree in the DIT. Each entry in the subtree is assigned the attribute(s) in this group attribute certificate.

16.4.2 Group role naming

In group role naming, the members of the group are identified by the attributes that they hold, such attributes being assigned to them in role assignment attribute certificates. In group role naming, the **holder** component of the group attribute certificate takes the **entityName** option and holds the role(s) of the group members who are being assigned the **attributes** in this group attribute certificate. The **GeneralNames** should contain a single **GeneralName** containing a **directoryName** with a single relative distinguished name (RDN), whose attribute type is the **role** attribute defined in clause 16.5.1. If **roleAuthority** in the **role** attribute is present, this identifies the attribute authorities who are responsible for issuing the role assignment certificates to holders who are members of this group. If **roleAuthority** is absent from the **role** attribute, the identity of the responsible AAs to issue the role assignment certificates shall be determined through means outside this Specification. The **roleName** component of the **role** attribute identifies the role(s) of the group who are being assigned the **attributes** in this group attribute certificate.

NOTE 1 – Group role naming allows attribute based role assignments, role mappings and role hierarchies to be defined, by specifying that members of other (more powerful) roles are assigned the roles of this group attribute certificate.

NOTE 2 – Where the role in the **holder** component is the same as the role in the **attributes** component of this group attribute certificate, this is delegation of authority from the issuer of the group attribute certificate to the **roleAuthority** in the **role** attribute. However, a much simpler way of achieving the same effect is to use the **roleAuthority** as the holder.

16.5 Roles model

Roles provide a means to indirectly assign privileges to individuals. Individuals are issued role assignment certificates that assign one or more roles to them through an instance the role attribute type contained in the certificate. Specific privileges are assigned to a role name through role specification certificates, rather than to individual privilege holders through attribute certificates. This level of indirection enables, for example, the privileges assigned to a role to be updated, without impacting the certificates that assign roles to individuals. Role assignment certificates may be attribute certificates or public-key certificates. Role specification certificates may be attribute certificates, but not public-key certificates. If role specification certificates are not used, the assignment of privileges to a role may be done through other means (e.g., may be locally configured at a privilege verifier).

The following are all possible:

– any number of roles can be defined by any AA;

– the role itself and the members of a role can be defined and administered separately, by different AAs;

– role membership, just as any other privilege, may be delegated; and

– roles and membership may be assigned any suitable lifetime.

If the role assignment certificate is an attribute certificate, the **role** attribute is contained in the **attributes** component of the attribute certificate. If the role assignment certificate is a public-key certificate, the **role** attribute is contained in the **subjectDirectoryAttributes** extension. In the latter case, any additional privileges contained in the public-key certificate are privileges that are directly assigned to the public-key certificate subject, not privileges assigned to the role.

Thus, a privilege asserter may present a role assignment certificate to the privilege verifier demonstrating only that the privilege asserter has a particular role (e.g., "manager", or "purchaser"). The privilege verifier may know *a priori*, or may have to discover by other means, the privileges associated with the asserted role in order to make a pass/fail authorization decision. The role specification attribute certificate can be used for this purpose.

A privilege verifier needs to have an understanding of the privileges specified for the role. The assignment of those privileges to the role may be done within the PMI in a role specification attribute certificate or outside the PMI (e.g., locally configured). If the role privileges are asserted in a role specification attribute certificate, mechanisms for linking that certificate with the relevant role assignment attribute certificate for the privilege asserter are provided in this Specification. A role specification attribute certificate cannot be delegated to any other entity. The issuer of the role assignment certificate may be independent of the issuer of the role specification certificate and these may be administered (expired, revoked, and so on) entirely separately. The same certificate (attribute certificate or public-key certificate) can be a role assignment certificate, as well as contain assignment of other privileges directly to the same individual. However, a role specification attribute certificate shall be a separate certificate.

NOTE – The use of roles within an authorization framework can increase the complexity of path processing, because such functionality essentially defines another delegation path which needs to be followed. The delegation path for the role assignment certificate may involve different AAs and may be independent of the AA that issued the role specification certificate.

16.5.1 Role attribute type

The specification of privilege attribute types is generally an application-specific issue that is outside the scope of this Specification. The single exception to this is an attribute defined here for the assignment of a holder to a role. The specification of values for the role attribute is outside the scope of this Specification.

```
role ATTRIBUTE ::= {
  WITH SYNTAX  RoleSyntax
  ID           id-at-role }

RoleSyntax ::= SEQUENCE {
  roleAuthority  [0]  GeneralNames OPTIONAL,
  roleName       [1]  GeneralName,
  ... }
```

An attribute of this privilege attribute type may be used to populate the **attributes** component of a role assignment certificate or to populate the **holder** component of a role specification or group attribute certificate, or both.

If the role assignment certificate is a public-key certificate rather than an attribute certificate, the **role** attribute may be used to populate the **subjectDirectoryAttributes** extension of that public-key certificate.

When the **role** attribute is used to populate the **attributes** component of a role assignment attribute certificate, the **roleAuthority**, if present, identifies the recognized authority that is responsible for issuing the role specification attribute certificate. If there are multiple occurrences of **GeneralName**, they shall all be alternative names for the same authority.

If **roleAuthority** is present, and a privilege verifier uses a role specification certificate to determine the privileges assigned to the role, at least one of the names in **roleAuthority** shall be present in the **issuer** component of that role specification attribute certificate. If the privilege verifier has used means other than a role specification attribute certificate to determine the privileges assigned to the role, mechanisms to ensure that those privileges were assigned by an authority named in this component are outside the scope of this Specification.

If **roleAuthority** is absent, the identity of the responsible authority shall be determined through other means. The **roleSpecCertIdentifier** extension in a role assignment certificate is one way to achieve this binding, in the case where a role specification certificate was used to assign privileges to the role.

The **roleName** component identifies the role to which the holder of this role assignment attribute certificate is assigned. If a privilege verifier uses a role specification certificate to determine the privileges assigned to that role, this role name shall also appear in the **holder** component of the role specification attribute certificate.

When the **role** attribute is used to populate the **holder** component of a group attribute certificate, the **roleAuthority**, if present, identifies the recognized authorities that are responsible for issuing role assignment certificates to holders who are members of the group being assigned the attributes in this group attribute certificate. If **roleAuthority** is absent, the identity of the responsible authorities to issue the role assignment certificates shall be determined through other means. The **roleName** component identifies the role(s) of the group of holders who are being assigned the attributes in this group attribute certificate. This **roleName** shall also appear in the **attributes** component of the role assignment certificates of the group of holders who are being assigned the attributes in this group attribute certificate. Where more than one role value is present in **roleName**, a group member must be assigned all the role values (in one or more role assignment certificates) in order to be assigned the attributes in this group attribute certificate.

When the **role** attribute is used to populate both the **holder** component and the **attributes** component, this is a role mapping attribute certificate.

16.6 Recognition of Authority Model

Figure 8 shows the control model for a single domain PMI.

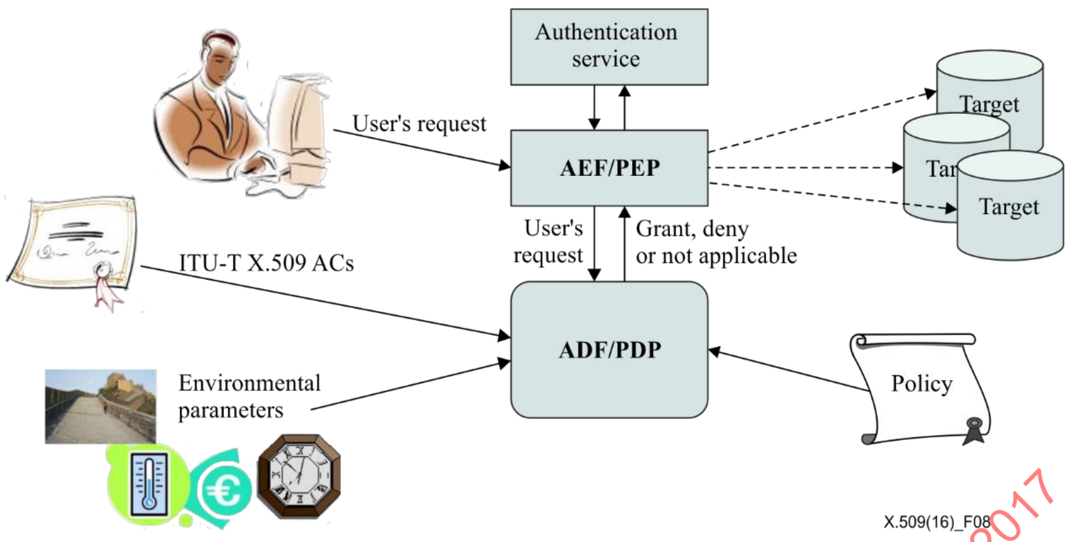

Figure 8 – The control model for a single domain PMI

The PMI policy contains information that directs the PDP in making its access control decisions. This information typically includes data about the trusted SOA, the delegation rules, which attributes are known and used, and which privileges are needed to gain access to which resources, etc. The policy information may be statically configured into the PDP, or may be dynamically obtained, for example, by passing a protected privilege policy attribute certificate to the PDP.

In order to support federations between organizations, and the construction of dynamic virtual organizations, it is essential that PMIs can be plugged together, so that attribute certificates issued in one domain can be used effectively in another PMI domain to gain access to its resources. Otherwise, the second PMI domain will have to issue another set of attribute certificates to the users of the first domain. This is both inefficient and cumbersome for the users to manage.

Recognition of Authority is the feature that will facilitate the rapid integration of PMIs from different domains into a single federated PMI.

In Figure 9, the user, who is a member of the TopLeft domain, wishes to access the resources of the BottomRight domain. He or she might contact the BottomRight domain directly, or his or her request may be relayed by the gatekeeper (AEF/PEP) in the TopLeft domain. Either way, the PDP in the BottomRight domain needs to understand the attribute certificates issued by the TopLeft domain, and the BottomRight policy needs to tell the BottomRight PDP whether they are sufficient to grant access to the requested resource or not.

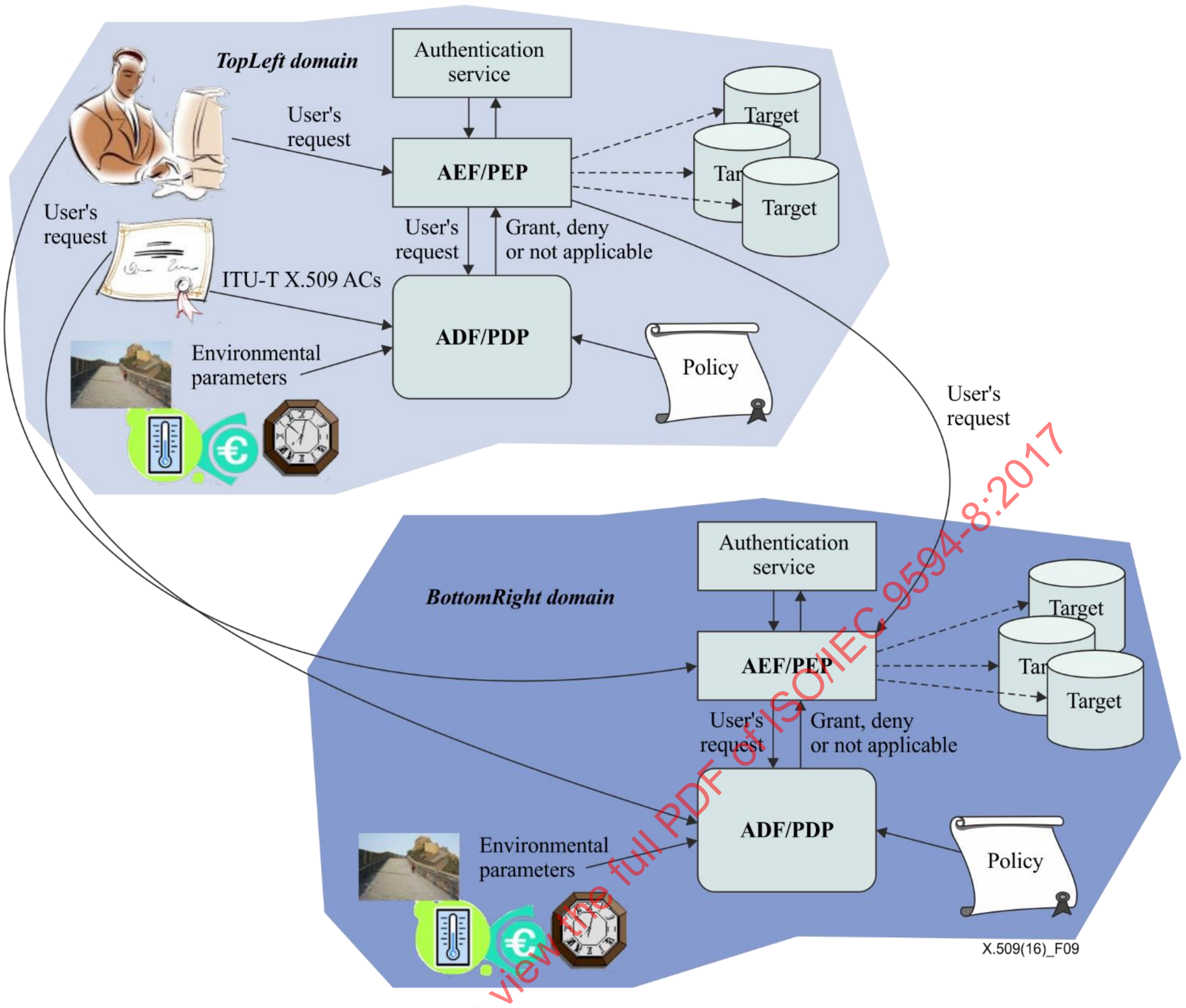

Figure 9 – Two federated PMI domains

The SOA in the trusting (local) domain (e.g., the BottomRight domain) needs to update its policy so that the SOA of the remote domain (e.g., the TopLeft domain) becomes trusted or recognized. The local policy can be updated in (at least) one of two ways:

a) statically, by adding extra information into the policy that is loaded into the local PDP prior to it making access control decisions;

b) dynamically, by issuing a new supplementary policy that adds additional information to the current policy. This dynamic addition to the local policy could be by the local SOA issuing a policy attribute certificate to the remote SOA or by the local SOA issuing an administrative role attribute certificate to the remote SOA so that the remote SOA may issue its own policy attribute certificate. In both cases, these need to be read in by the local PDP prior to validating a request from a user of the remote domain.

When the local SOA issues a policy attribute certificate to the remote SOA, it may be as follows:

– the **`holder`** component identifies the SOA of the remote domain;

– the **`issuer`** component identifies the local SOA;

– the attributes of the attribute certificate are the union of all the privilege attributes that the remote SOA is trusted to issue. If any of these privilege attributes are newly defined roles, then new role specification attribute certificates may also need to be issued;

– **`basicAttConstraints`** extension is included with **`authority`** set to **`TRUE`** to indicate that the remote SOA is an AA. Path length constraint (**`pathLenConstraint`**) is set as appropriate to indicate the length of the delegation chain that is allowed in the remote domain;

– **`holderNameConstraints`** may be set to limit the name forms and namespaces in which the remote SOA can assign privilege attributes to users;

– **`allowedAttributeAssignments`** may be set to further constrain which groups of remote holders can be assigned which sets of privilege attributes;
– **`attributeMappings`** may be set to inform the local PDPs which remotely assigned attributes should be considered equal to which locally assigned attributes.

When the local SOA issues an administrative role attribute certificate to the remote SOA, it may work as follows:

1) The local SOA defines an administrative role for the local domain and the permissions that may be administered by this administrative role. This may be defined in a role specification attribute certificate in which the holder is the administrative role and the attribute is the **`permission`** attribute (defined in clause 16.8.1 below). The set of permissions for an administrative role is called the administrative scope of an administrative role. These permissions may also be assigned to local roles, so that users with these local roles will inherit these permissions. Issuing an administrative role specification attribute certificate allows remote administrators to learn their administrative scope.
2) The local SOA delegates this administrative role to the remote SOA by issuing a role-assignment attribute certificate to the remote SOA containing the assigned administrative role. The remote SOA may also be allowed to delegate the administrative role to other administrators in the remote domain, as determined by **`pathLenConstraint`** in the **`basicAttConstraints`** extension in the role-assignment attribute certificate.
3) The remote SOA (or subordinate AA) that has been assigned this administrative role is now recognized as an entity able to issue two types of delegated-policy attribute certificate, either a delegated-role specification attribute certificate or a delegated attribute mapping attribute certificate. In a delegated-role-specification attribute certificate, the remote SOA (or AA) directly assigns the permissions from the administrative scope to new remotely defined attributes as described below. In a delegated attribute mapping attribute certificate, new remotely defined attributes are mapped into existing local roles as described below.
4) In order to ensure that the remote SOA (or AA) cannot overstep its delegated authority, the authorization system has to validate that the privileges stated or implied by a delegated policy attribute certificate lie within the administrative scope defined for the administrative role. If they do, the delegated policy attribute certificate is accepted, and its policy rules become dynamically incorporated into the local SOA's policy. If they do not, the delegated policy attribute certificate is rejected, and its policy rules will be ignored.

A delegated role specification attribute certificate comprises:

– the holder is the newly specified remote role;
– the issuer component identifies the remote SOA (or AA) of the remote domain that issued this attribute certificate;
– the attributes of the attribute certificate are the privileges that will be assigned to users in the remote domain who are assigned the remote role;
– **`holderNameConstraints`** may be set to limit the name forms and namespaces of the users which may be assigned these privilege attributes;
– **`allowedAttributeAssignments`** may be set to further constrain which groups of remote holders can be assigned which sets of remotely defined privilege attributes.

A delegated attribute mapping policy attribute certificate comprises:

– the **`holder`** and the **`issuer`** components identify the remote SOA (or AA) of the remote domain that issued this attribute certificate;
– the **`attributes`** component is null;
– **`holderNameConstraints`** may be set to limit the name forms and namespaces of the users which may be assigned these privilege attributes;
– **`allowedAttributeAssignments`** may be set to further constrain which groups of remote holders can be assigned which sets of privilege attributes;
– **`attributeMappings`** is set to inform the PDP which remotely assigned attributes should be considered equal to which locally assigned attributes.

The remote SOA will subsequently issue privilege attribute within attribute certificates to end users and/or to AAs in its domain. Whether the remote AAs are trusted or not, and if trusted, the number of AAs that are allowed in a delegation chain, may be set by the **`pathLenConstraint`** in the attribute certificate issued to the remote SOA. The privilege attributes in the attribute certificates issued by the remote SOA may contain either:

– permissions that are understood by the PDPs in the local domain; or
– roles which may or may not be understood by the PDPs in the local domain.

When an attribute certificate contains roles that are not understood by the local PDPs, the latter must know how to map these unknown roles into local permissions. This can be achieved in at least one of four ways. If the local SOA knows what these roles are likely to be prior to recognizing the remote SOA, then if it issues a policy attribute certificate to the remote SOA an attribute mapping extension can be placed in the policy attribute certificate issued to the remote SOA, or alternatively attribute mapping rules can be added into the policy loaded by the local PDP. If the remote roles are not known prior to recognizing the remote SOA, the remote SOA will need to either issue an attribute mapping policy attribute certificate or place the attribute mapping extension in the attribute certificates that it issues to its users.

If the remote SOA issues an attribute mapping policy attribute certificate, this should contain:

– a holder and issuer name which is that of the remote SOA;

– the **`attributes`** component shall be an empty sequence;

– **`attributeMappings`** extension set to describe the attribute mappings.

NOTE – A remote SOA should not issue an attribute mapping attribute certificate in which both the holder and attributes are roles, since this type of attribute mapping should be issued by the local SOA only.

This attribute mapping policy attribute certificate needs to be made available to the local PDPs at decision time. This can be done by either storing the policy attribute certificate in the directory entry of the remote SOA and giving the local PDPs read access to it (the pull model) or by including the policy attribute certificate in the set of attribute certificates presented by the remote user when accessing the local resource (the push model).

16.7 XML privilege information attribute

The specification of privileges is generally an application-specific issue that is outside the scope of this Specification. While this attribute does not define any specific privilege information, it provides a container attribute in which XML-encoded privileges can be conveyed in attribute certificates.

```
xmlPrivilegeInfo ATTRIBUTE ::= {
  WITH SYNTAX  UTF8String --contains XML-encoded privilege information
  ID           id-at-xMLPrivilegeInfo }
```

The XML schema for the role attribute type can be defined either with ASN.1 or with XML Schema Definition (XSD).

The XML contained within the **`UTF8String`** needs to be self-identifying.

The following is an ASN.1 schema defining an XML role attribute type. It is followed by an XSD specification for the same attribute type, and by an example XML instance. The example instance is a valid instance for both the ASN.1 and the XSD schema instances, and can be validated by either ASN.1 or XSD tools.

The example schema defines a role attribute with an ID, an issuing authority and the name of the role.

```
CERTIFICATE-ATTRIBUTE DEFINITIONS ::=
BEGIN
Role ::= [UNCAPITALIZED] SEQUENCE {
  id            [ATTRIBUTE] XML-ID,
  authorities   SEQUENCE (1..MAX) OF
    authority     UTF8String,
    name          UTF8String }

XML-ID ::= UTF8String
END
```

The following XSD schema is an alternative (exactly equivalent) definition:

```
<schema xmlns="http://www.w3.org/2000/08/XMLSchema">
  <element name="role">
    <attribute name="id" type="ID"/>
      complexType>
        <sequence>
          <element name="authorities">
            <complexType>
              <sequence>
                <element name="authority" type="string" minOccurs="1" maxOccurs="*"/>
             </sequence>
            </complexType>
         </element>
         <element name="name" type="string"/>
        </sequence>
      </complexType>
  </element>
</schema>
```

An example of an instance conforming to the above schema definitions, that would be a value of the **xMLPrivilegeInfo** attribute type would be:

```
<role id="123" xmlns="http://www.example.org/certificates/attribute">
  <authorities>
    <authority>Fictitious Organization</authority>
  </authorities>
  <name>manager</name>
</role>
```

16.8 Permission attribute and matching rule

16.8.1 Permission attribute

This attribute defines a general permission, which is an operation on an object, e.g., a read operation on a file object. The specification of values for the operations or objects is outside the scope of this Specification. Note that the names of both operations and objects are case sensitive.

```
permission ATTRIBUTE ::= {
  WITH SYNTAX              DualStringSyntax
  EQUALITY MATCHING RULE   dualStringMatch
  ID                       id-at-permission }

DualStringSyntax ::= SEQUENCE {
  operation   [0]  UnboundedDirectoryString,
  object      [1]  UnboundedDirectoryString,
  ... }
```

An attribute of the **permission** attribute type is intended to be used to populate the **attributes** component of an attribute certificate and is not intended for storing as an attribute of a directory entry.

16.8.2 Dual string matching rule

The **dualStringMatch** matching rule is a case sensitive matching rule and is defined as follows:

```
dualStringMatch MATCHING-RULE ::= {
  SYNTAX  DualStringSyntax
  ID      id-mr-dualStringMatch }
```

The **dualStringMatch** matching rule performs a case sensitive comparison for equality between a pair of presented strings and an attribute value of type **DualStringSyntax**, in which the first presented string is the operation and the second presented string is the object.

17 Attribute certificate and attribute certificate revocation list extensions

The following certificate extensions may be included in certificates for the purposes of privilege management. Along with the definition of the extensions themselves, the rules for certificate types in which the extension may be present are also provided.

With the exception of the SOA identifier extension, any of the extensions that may be included in a public-key certificate shall only be included if that public-key certificate is one that assigns privilege to its subject (i.e., the **subjectDirectoryAttributes** extension shall be present). If any of these extensions is present in a public-key certificate, that extension applies to all privileges present in the **subjectDirectoryAttributes** extension.

Revocation lists used to publish revocation notices for attribute certificates may contain any CRL or CRL entry extension as defined for use in CRLs in Section 2 of this Specification. There are also public-key certificate extensions that are applicable for attribute certificates. How such extensions are to be included in attribute certificates and ACRLs, is specified in this clause.

This clause specifies extensions in the following areas:

a) Basic privilege management: These certificate extensions convey information relevant to the assertion of a privilege.

b) Privilege revocation: These certificate extensions convey information regarding the location of revocation status information.

c) Source of authority (SOA): These certificate extensions relate to the trusted source of privilege assignment by a verifier for a given resource.

d) Roles: These certificate extensions convey information regarding the location of related role specification certificates.

e) Delegation: These certificate extensions allow constraints to be set on the subsequent delegation of assigned privileges.

f) Recognition of authority: These certificate extensions allow PMIs to be federated together.

17.1 Basic privilege management extensions

17.1.1 Requirements

The following requirements relate to basic privilege management:

a) Issuers need to be able to place constraints on the time during which a privilege can be asserted.

b) Issuers need to be able to target attribute certificates to specific servers/services.

c) It may be necessary for issuers to convey information intended for display to privilege asserters and/or privilege verifiers using the certificate.

d) Issuers may need to be able to place constraints on the privilege policies with which the assigned privilege can be used.

e) Issuers may need to be able to issue an attribute certificate that can only be asserted once within its lifetime.

f) Issuers may need to be able to issue privilege attributes to a group of entities that share a common property.

17.1.2 Basic privilege management extension

The following extensions are defined:

a) Time specification;

b) Targeting information;

c) User notice;

d) Acceptable privilege policies;

e) Indirect issuer;

f) Single use;

g) Group attribute certificate.

17.1.2.1 Time specification extension

17.1.2.1.1 Time specification extension definition

The time specification extension can be used by an AA to restrict the specific periods of time during which the privilege, assigned in the certificate containing this extension, can be asserted by the privilege holder. For example, an AA may issue a certificate assigning privileges which can only be asserted between Monday and Friday and between the hours of 9:00 a.m. and 5:00 p.m.. Another example, in the case of delegation, might be a manager delegating the signing authority to a subordinate for the time that the manager will be away on vacation.

This extension is defined as follows:

```
timeSpecification EXTENSION ::= {
  SYNTAX         TimeSpecification
  IDENTIFIED BY  id-ce-timeSpecification }
```

This extension may be present in attribute certificates or in public-key certificates with the **subjectDirectoryAttributes** extension issued to entities that may act as privilege asserters. This extension shall not be included in public-key certificates that contain the **sOAIdentifier** extension or in attribute certificates issued to entities acting as AAs that may not also act as privilege asserters.

If this extension is present in an attribute certificate issued to an entity acting as an AA, it applies only to that entity's assertion of the privileges contained in the attribute certificate. It does not impact the time period during which the AA is able to issue attribute certificates.

Because this extension is effectively specifying a refinement on the validity period of the certificate that contains it, this extension shall be flagged as critical (i.e., the issuer, by including this extension, is explicitly defining the privilege assignment to be invalid outside the time specified).

If this extension is present, but not understood by the privilege verifier, the certificate shall be considered invalid.

17.1.2.1.2 Time specification matching rule

The time specification matching rule compares for equality a presented value with an attribute value of type **AttributeCertificate**.

```
timeSpecificationMatch MATCHING-RULE ::= {
  SYNTAX  TimeSpecification
  ID      id-mr-timeSpecMatch }
```

This matching rule returns TRUE if the attribute/public-key certificate as stored in a directory contains the **timeSpecification** extension and if components that are present in the presented value match the corresponding components of the stored attribute/public-key certificate **timeSpecification** extension.

17.1.2.2 Targeting information extension

The targeting information extension enables the targeting of an attribute certificate to a specific set of servers/services. An attribute certificate that contains this extension should only be usable at the specified servers/services.

This extension is defined as follows.

```
targetingInformation EXTENSION ::= {
  SYNTAX         SEQUENCE SIZE (1..MAX) OF Targets
  IDENTIFIED BY  id-ce-targetingInformation }

Targets ::= SEQUENCE SIZE (1..MAX) OF Target

Target ::= CHOICE {
  targetName   [0]  GeneralName,
  targetGroup  [1]  GeneralName,
  targetCert   [2]  TargetCert,
  ... }

TargetCert ::= SEQUENCE {
  targetCertificate  IssuerSerial,
  targetName         GeneralName OPTIONAL,
  certDigestInfo     ObjectDigestInfo OPTIONAL }
```

The **Target** data type has the following three alternatives:

a) The **targetName** alternative shall provide the name of target servers/services for which the containing attribute certificate is targeted.

b) The **targetGroup** alternative shall provide the name of a target group for which the containing attribute certificate is targeted. How the membership of a target within a **targetGroup** is determined is outside the scope of this Specification.

c) The **targetCert** alternative shall identify target servers/services by reference to their certificate.

This extension may be present in attribute certificates issued by AAs, including SOAs, to entities that may act as privilege asserters, including other AAs and end entities. This extension shall not be included in public-key certificates or in attribute certificates issued to AAs that may not also act as privilege asserters.

If this extension is present in an attribute certificate issued to an entity that is an AA, it applies only to that entity's assertion of the privileges contained in the attribute certificate. It does not impact the AA ability to issue certificates.

This extension shall always be flagged as critical.

If this extension is present, but the privilege verifier is not among those specified, the attribute certificate should be considered invalid.

If this extension is not present, then the attribute certificate is not targeted and may be accepted by any server.

17.1.2.3 User notice extension

The user notice extension enables an AA to include a notice that should be displayed to the holder, when asserting their privilege, and/or to a privilege verifier when making use of the attribute certificate containing this extension.

This extension is defined as follows:

```
userNotice EXTENSION ::= {
  SYNTAX         SEQUENCE SIZE (1..MAX) OF UserNotice
  IDENTIFIED BY  id-ce-userNotice }
```

This extension may be present in attribute certificates or in public-key certificates with the **subjectDirectoryAttributes** extension issued to entities that may act as privilege asserters. This extension shall not be included in public-key certificates that contain the **sOAIdentifier** extension or in attribute certificates issued to entities acting as AAs that may not also act as privilege asserters.

If this extension is present in a certificate issued to an entity that is an AA, it applies only to that entity's assertion of the privileges contained in the certificate. It does not impact the AA ability to issue attribute certificates.

This extension may, at the option of the certificate issuer, be either flagged as critical or as non-critical.

If this extension is flagged as critical, the user notices shall be displayed to a privilege verifier each time a privilege is asserted. If the privilege asserter supplies the attribute certificate to the privilege verifier (i.e., the privilege verifier does not retrieve it directly from a repository), the user notices shall also be displayed to the privilege asserter.

If this extension is flagged as non-critical, the privilege asserted in the certificate may be granted by a privilege verifier regardless of whether or not the user notices were displayed to the privilege asserter and/or privilege verifier.

The **UserNotice** data type is defined in IETF RFC 5280 and provided here for easy reference:

```
UserNotice ::= SEQUENCE {
  noticeRef     NoticeReference OPTIONAL,
  explicitText  DisplayText OPTIONAL }

NoticeReference ::= SEQUENCE {
  organization   DisplayText,
  noticeNumbers  SEQUENCE OF INTEGER }

DisplayText ::= CHOICE {
  visibleString  VisibleString(SIZE (1..200)),
  bmpString      BMPString(SIZE (1..200)),
  utf8String     UTF8String(SIZE (1..200)) }
```

17.1.2.4 Acceptable privilege policies extension

The acceptable privilege policies extension is used to constrain the assertion of the assigned privileges for use with a specific set of privilege policies.

This extension is defined as follows:

```
acceptablePrivilegePolicies EXTENSION ::= {
  SYNTAX         AcceptablePrivilegePoliciesSyntax
  IDENTIFIED BY  id-ce-acceptablePrivilegePolicies }

AcceptablePrivilegePoliciesSyntax ::= SEQUENCE SIZE (1..MAX) OF PrivilegePolicy
```

This extension may be present in attribute certificates or in public-key certificates with the **subjectDirectoryAttributes** extension issued to entities that may act as privilege asserters. If this extension is contained in a public-key certificate it relates only to the subject's ability to act as a privilege asserter for the privileges contained in the **subjectDirectoryAttributes** extension.

If present, this extension shall be flagged as critical.

If this extension is present and the privilege verifier understands it, the privilege verifier shall ensure that the privilege policy that these privileges are being compared to is one of those identified in this extension.

If this extension is present, but not understood by the privilege verifier, the certificate shall be considered invalid.

17.1.2.5 Single use extension

In some scenarios, an AA may wish to issue an attribute certificate that can only be asserted once to a privilege verifier within the lifetime of the attribute certificate. The **singleUse** extension is defined as follows:

```
singleUse EXTENSION ::= {
  SYNTAX         NULL
  IDENTIFIED BY  id-ce-singleUse }
```

This extension may be present in end-entity attribute certificates issued by AAs and SOAs. This extension shall not be included in public-key certificates or in attribute certificates issued to AAs.

This extension shall always be flagged as critical.

Any privilege verifier that accepts a **`singleUse`** attribute certificate should keep a record of at least the issuer and serial number of the attribute certificate, until after the expiry date of the attribute certificate in order to ensure that the holder cannot use the attribute certificate again. Ideally, all privilege verifiers for which the attribute certificate is valid should have a coordination capability to ensure that the holder is not able to use the **`singleUse`** certificate with multiple privilege verifiers. Alternatively, the issuer of the **`singleUse`** attribute certificate should include a **`targetingInformation`** extension in the attribute certificate to limit the privilege verifier at which the attribute certificate is valid.

17.1.2.6 Group attribute certificate extension

In some scenarios it might be required for an AA to issue an attribute certificate to a group of entities that share a common property, for example, a set of web servers or a team of people, rather than to a single entity. Each group attribute certificate may be flagged as such by adding the group attribute certificate extension into the attribute certificate.

```
groupAC EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY  id-ce-groupAC }
```

This extension may be flagged as critical or as non-critical. This extension shall only be added to end-entity attribute certificates, and not to AA attribute certificates or to public-key certificates.

17.1.2.7 Authority key identifier extension

17.2 Privilege revocation extensions

17.2.1 Requirements

The following requirements relate to the revocation of attribute certificates:

a) In order to control CRL sizes, it may be necessary to assign subsets of the set of all certificates issued by one AA to different CRLs.

b) Attribute certificate issuers need to be able to indicate, in an attribute certificate, that no revocation information is available for that certificate.

17.2.2 Privilege revocation extensions

The following extensions are defined or referenced:

a) CRL distribution points;

b) No revocation information.

17.2.2.1 Use of CRL distribution points extension

The CRL distribution points extension is defined in clause 9.6.2.1 for use in public-key certificates. This extension may also be included in an attribute certificate. It may be present in attribute certificates issued to entities acting as AAs, including SOAs, as well as attribute certificates issued to entities acting ad PMI end entities.

If present in an attribute certificate, a privilege verifier shall process this extension in exactly the same manner as described for a relying party in clause 9.6.2.1 for public-key certificates by substituting public-key certificate, CA and CRL by attribute certificate, AA and ACRL, respectively.

17.2.2.2 AA issuing distribution point extension

This ACRL extension identifies the ACRL distribution point for attribute certificates for this particular ACRL, and indicates if the ACRL is indirect or if it is limited to covering only a subset of the revocation information. The limitation may be based on a subset of the attribute certificate population or on a subset of revocation reasons. The ACRL is signed by the ACRL issuer's private key – ACRL distribution points do not have their own key pairs. However, for an ACRL distributed via a directory, the ACRL is stored in the entry of the CRL distribution point, which may not be the directory entry of the ACRL issuer. If the issuing distribution point extension, the AA issuing distribution point extension, and the CRL scope extension are all absent, the ACRL shall contain entries for all revoked unexpired attribute certificates issued by the ACRL issuer.

After an attribute certificate appears on an ACRL, it may be deleted from a subsequent ACRL after the attribute certificate's expiry.

This extension is defined as follows:

```
aAissuingDistributionPoint EXTENSION ::= {
  SYNTAX          AAIssuingDistPointSyntax
  IDENTIFIED BY   id-ce-aAissuingDistributionPoint }

AAIssuingDistPointSyntax ::= SEQUENCE {
  distributionPoint            [0]  DistributionPointName OPTIONAL,
  onlySomeReasons              [1]  ReasonFlags OPTIONAL,
  indirectCRL                  [2]  BOOLEAN DEFAULT FALSE,
  containsUserAttributeCerts   [3]  BOOLEAN DEFAULT TRUE,
  containsAACerts              [4]  BOOLEAN DEFAULT TRUE,
  containsSOAPublicKeyCerts    [5]  BOOLEAN DEFAULT TRUE,
  ... }
```

The **distributionPoint** component contains the name of the distribution point in one or more name forms. If **onlySomeReasons** is present, the ACRL only contains revocations for attribute certificates for the identified reason or reasons; otherwise, the ACRL contains revocations for all reasons.

If **indirectCRL** is **TRUE**, then the ACRL may contain revocation notifications for attribute certificates from AAs other than the issuer of the ACRL. The particular AA responsible for each entry is as indicated by the certificate issuer ACRL entry extension in that entry or in accordance with the defaulting rules described in clause 17.2.2.3. In such an ACRL, it is the responsibility of the ACRL issuer to ensure that the ACRL is complete in that it contains all revocation entries, consistent with **containsUserAttributeCerts**, **containsAACerts**, **containsSOAPublicKeyCerts** and **onlySomeReasons** indicators, from all AAs that identify this ACRL issuer in their attribute certificates.

If **containsUserAttributeCerts** is **TRUE**, the ACRL contains revocations for end-entity attribute certificates. If **containsAACerts** is **TRUE**, the ACRL contains revocations for attribute certificates issued to subjects that are themselves AAs.

If **containsSOAPublicKeyCerts** is **TRUE**, the ACRL contains revocations for public-key certificates issued to an entity that is an SOA for the purposes of privilege management (i.e., attribute certificates that contain the **SOAIdentifier** extension). For ACRLs distributed via a directory, the following rules apply. If the ACRL is a dACRL, it shall be distributed via the **deltaRevocationList** attribute of the associated distribution point or, if no distribution point is identified, via an attribute of type **aDeltaRevocationList** of the ACRL issuer entry, regardless of the settings for certificate types covered by the ACRL. Unless the ACRL is a dACRL:

– an ACRL that does not contain an **issuingDistributionPoint** extension which has only **containsAACerts** and/or **containsSOAPublicKeyCerts** set to **TRUE** shall be distributed via the **attributeAuthorityRevocationList** attribute of the associated distribution point or, if no distribution point is identified, via the **attributeAuthorityRevocationList** attribute of the CRL issuer entry;

– an ACRL that does not contain an **issuingDistributionPoint** extension which has **containsUserAttributeCerts** set to **TRUE** (with or without **containsAACerts** and/or **containsSOAPublicKeyCerts** also set) shall be distributed via the **attributeCertificateRevocationList** attribute of the associated distribution point or, if no distribution point is identified, via the **attributeCertificateRevocationList** attribute of the ACRL issuer entry;

– an ACRL which contains an **issuingDistributionPoint** extension shall be distributed as specified in clause 9.6.2.2.

This extension shall always be flagged as critical. A privilege verifier that does not understand this extension cannot assume that the ACRL contains a complete list of revoked certificates of the identified authority. CRLs not containing critical extensions shall contain all current CRL entries for the issuing authority, including entries for all revoked end-entity certificates and authority certificates.

NOTE – The means by which revocation information is communicated by authorities to ACRL issuers is beyond the scope of this Specification.

If an authority publishes a ACRL with the **containsAACerts** component set to TRUE and the **containsUserAttributeCerts** not set to TRUE, then the authority shall ensure that all AA certificates covered by this CRL contain the **basicAttConstraints** extension.

If an authority publishes an ACRL with contains **SOAPublicKeyCerts** set to **TRUE**, then the authority shall ensure that all SOA certificates covered by this ACRL contain the **SOAIdentifier** extension.

17.2.2.3 Use of certificate issuer extension

The certificate issuer extension is defined in clause 9.6.2.3 for use in indirect CRLs. This extension may also be included in indirect ACRLs.

The procedure in clause 9.6.2.3 applies by replacing issuing distribution point extension, CRL and CA with AA issuing distribution point extension, ACRL and AA, respectively.

17.2.2.4 Use of delta CRL indicator extension

The delta CRL indicator extension is defined in clause 9.6.2.4 for use in a dCRL to an CRL and may also be used in a dACRL to an ACRL.

The procedure specified in clause 9.6.2.4 shall be followed by replacing public-key certificate, CA, CRL and dCRL with attribute certificate, AA, ACRL and dACRL, respectively.

17.2.2.5 Use of base update extension

This extension shall be used as specified in clause 9.6.2.5 by replacing dCRLs with dACRLs.

17.2.2.6 Use of freshest CRL extension

The **`freshest`** CRL extension is defined in clause 9.6.2.6 for use in a dCRL. It may also be used for a dACRL.

The procedure specified in clause 9.6.2.6 shall be followed by replacing public-key certificate, CA, CRL, dCRL and relying party with attribute certificate, AA, ACRL, dACRL and privilege verifier, respectively.

17.2.2.7 No revocation information available extension

In some environments (e.g., where attribute certificates are issued with very short validity periods), there may not be a need to revoke attribute certificates. An AA may use this extension to indicate that revocation status information is not provided for this attribute certificate. This extension is defined as follows:

```
noRevAvail EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY   id-ce-noRevAvail }
```

This extension may be present in attribute certificates issued by AAs, including SOAs, to entities acting as PMI end entities. This extension shall not be included in public-key certificates or in attribute certificates issued to AAs.

This extension shall always be flagged as non-critical.

If this extension is present in an attribute certificate, a privilege verifier need not seek revocation status information.

17.3 Source of authority extensions

17.3.1 Requirements

The following requirements relate to SOAs:

a) In some environments there is a need for tight control by a CA, of the entities that can act as SOAs.

b) There is a need to make the valid syntax definitions and domination rules for privilege attributes available by the responsible SOAs.

17.3.2 SOA extensions

The following extensions are defined:

a) SOA identifier;

b) Attribute descriptor.

17.3.2.1 SOA identifier extension

17.3.2.1.1 SOA identifier extension definition

This extension may only be present in a public-key certificate issued to an SOA. It shall not be included in attribute certificates or public-key certificates issued to other AAs.

The SOA identifier extension indicates that the public-key certificate subject may act as an SOA for the purposes of privilege management. As such, the public-key certificate subject may define attributes that assign privilege, issue attribute descriptor certificates for those attributes and use the private key corresponding to the certified public key to issue attribute certificates that assign privileges to holders. If the public key certificate is a CA certificate, the subject of that CA certificate may also issue public-key certificates with a **`subjectDirectoryAttributes`** extension containing the privileges.

In some environments, this extension is not required and other mechanisms may be used to determine the entities that may act as SOAs. This extension is required only in environments where tight centralized control by a CA is required to manage the entities that act as SOAs.

This extension is defined as follows:

```
sOAIdentifier EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY  id-ce-sOAIdentifier }
```

If this extension is not present in a certificate, the subject/holder ability to act as an SOA shall be determined by other means.

Cross-certification applies only to public-key certificates and not to attribute certificates. Therefore, a cross-certificate issued to the CA that is the issuer of a certificate containing the SOA identifier extension does not provide transitive trust to the SOA identified in this extension.

This extension shall always be flagged as non-critical.

17.3.2.1.2 SOA identifier matching rule

The SOA identifier matching rule compares a presented value with an attribute value of type **`Certificate`**.

```
sOAIdentifierMatch MATCHING-RULE ::= {
  SYNTAX  NULL
  ID      id-mr-sOAIdentifierMatch }
```

This matching rule returns TRUE if the stored public-key certificate contains a **`sOAIdentifier`** extension.

17.3.2.2 Attribute descriptor extension

17.3.2.2.1 Attribute descriptor extension definition

The definition of a privilege attribute, and the domination rules governing the subsequent delegation of that privilege, are needed by privilege verifiers to ensure that authorization is done correctly. These definitions and rules may be provided to privilege verifiers in a variety of ways outside the scope of this Specification (e.g., they may be locally configured at the privilege verifier).

This extension provides one mechanism that can be used by an SOA to make privilege attribute definitions and associated domination rules available to privilege verifiers. An attribute certificate that contains this extension is called an attribute descriptor certificate and is a special type of attribute certificate. Although syntactically identical to an **`AttributeCertificate`**, an attribute descriptor certificate:

- contains an empty **`SEQUENCE`** in its **`attributes`** component;
- is a self-issued certificate (i.e., the issuer and holder are the same entity); and
- includes the attribute descriptor extension.

This extension is defined as follows:

```
attributeDescriptor EXTENSION ::= {
  SYNTAX          AttributeDescriptorSyntax
  IDENTIFIED BY  {id-ce-attributeDescriptor} }

AttributeDescriptorSyntax ::= SEQUENCE {
  identifier              AttributeIdentifier,
  attributeSyntax         OCTET STRING(SIZE (1..MAX)),
  name               [0]  AttributeName OPTIONAL,
  description        [1]  AttributeDescription OPTIONAL,
  dominationRule          PrivilegePolicyIdentifier,
  ... }

AttributeIdentifier ::= ATTRIBUTE.&id({AttributeIDs})

AttributeIDs ATTRIBUTE ::= {...}

AttributeName ::= UTF8String(SIZE (1..MAX))

AttributeDescription ::= UTF8String(SIZE (1..MAX))

PrivilegePolicyIdentifier ::= SEQUENCE {
  privilegePolicy  PrivilegePolicy,
  privPolSyntax     InfoSyntax,
  ... }
```

The **identifier** component of a value of the **attributeDescriptor** extension is the object identifier identifying the attribute type.

The **attributeSyntax** component contains the ASN.1 definition of the attribute's syntax. Such an ASN.1 definition shall be given as specified for the information component of the Matching Rules operational attribute defined in Rec. ITU-T X.501 | ISO/IEC 9594-2.

The **name** component optionally contains a user-friendly name by which the attribute may be recognized.

The **description** component optionally contains a user-friendly description of the attribute.

The **dominationRule** component specifies, for the attribute, what it means for a delegated privilege to be "less than" the corresponding privilege held by the delegator.

– The **privilegePolicy** component identifies the instance of privilege policy that contains the rules, by its object identifier.

– The **privPolSyntax** component contains either the privilege policy itself or a pointer to a location where it can be located. If a pointer is included, an optional hash of the privilege policy can also be included to allow an integrity check on the referenced privilege policy.

This extension may only be present in attribute descriptor certificates. This extension shall not be present in public-key certificates or in attribute certificates other than self-issued certificates of SOAs.

This extension shall always be flagged as non-critical.

The attribute descriptor certificate, created by the SOA at the time of creation/definition of the corresponding attribute type, is a means by which the universal constraint of delegating "down" can be understood and enforced in the infrastructure. In a directory, attribute certificates that contain this extension would be stored in the **attributeDescriptorCertificate** attribute of the SOA's directory entry.

17.3.2.2.2 Attribute descriptor matching rule

The attribute descriptor matching rule compares for equality a presented value with an attribute value of type **AttributeCertificate.**

```
attDescriptor MATCHING-RULE ::= {
  SYNTAX   AttributeDescriptorSyntax
  ID       id-mr-attDescriptorMatch }
```

This matching rule returns TRUE if the stored value contains the **attributeDescriptor** extension and if components that are present in the presented value match the corresponding components of the stored value.

17.4 Role extensions

17.4.1 Requirements

The following requirement relates to roles:

– If a certificate is a role assignment certificate, a privilege verifier needs to be able to locate the corresponding role specification certificate that contains the specific privileges assigned to the role itself.

17.4.2 Role extensions

The following extension is defined:

– Role specification certificate identifier.

17.4.2.1 Role specification certificate identifier extension

17.4.2.1.1 Role specification certificate identifier extension definition

This extension may be used by an AA as a pointer to a role specification certificate that contains the assignment of privileges to a role. It may be present in a role assignment certificate (i.e., a certificate that contains the **role** attribute).

A privilege verifier, when dealing with a role assignment certificate, needs to obtain the set of privileges of that role in order to determine whether to pass or fail the verification. If the privileges were assigned to the role in a role specification certificate, this extension may be used to locate that certificate.

This extension is defined as follows:

```
roleSpecCertIdentifier EXTENSION ::= {
  SYNTAX         RoleSpecCertIdentifierSyntax
  IDENTIFIED BY  {id-ce-roleSpecCertIdentifier} }

RoleSpecCertIdentifierSyntax ::=
  SEQUENCE SIZE (1..MAX) OF RoleSpecCertIdentifier

RoleSpecCertIdentifier ::= SEQUENCE {
  roleName              [0]  GeneralName,
  roleCertIssuer        [1]  GeneralName,
  roleCertSerialNumber  [2]  CertificateSerialNumber OPTIONAL,
  roleCertLocator       [3]  GeneralNames OPTIONAL,
  ... }
```

The **roleName** identifies the role. This name would be the same as that in the **holder** component of the role specification certificate being referenced by this extension.

The **roleCertIssuer** identifies the AA that issued the referenced role specification certificate.

The **roleCertSerialNumber**, if present, contains the serial number of the role specification certificate. Note that if the privileges assigned to the role itself change, then a new role specification certificate would be issued to the role. Any certificates that contain this extension, including the **roleCertSerialNumber** component, would then need to be replaced by certificates that referenced the new serial number. Although this behaviour is required in some environments, it is undesirable in many others. Typically, this component would be absent, enabling automatic updating of the privileges assigned to the role itself, without impacting the role assignment certificates.

The **roleCertLocator**, if present, contains information that can be used to locate the role specification certificate.

This extension may be present in role assignment certificates that are attribute certificates or public-key certificates issued by AAs, including SOAs, to other AAs or to end-entity privilege holders. This extension shall not be included in certificates that contain the SOA identifier extension.

If present, this extension can be used by a privilege verifier to locate the role specification certificate.

If this extension is not present, either:

a) other means will be used to locate the role specification certificate; or

b) mechanisms other than a role specification certificate were used to assign privileges to the role (e.g., role privileges may be locally configured at the privilege verifier).

This extension shall always be flagged as non-critical.

17.4.2.1.2 Role specification certificate ID matching rule

The role specification certificate identifier matching rule compares for equality a presented value with an attribute value of type **AttributeCertificate**.

```
roleSpecCertIdMatch MATCHING-RULE ::= {
  SYNTAX  RoleSpecCertIdentifierSyntax
  ID      id-mr-roleSpecCertIdMatch }
```

This matching rule returns TRUE if the stored value contains the **roleSpecCertIdentifier** extension and if components that are present in the presented value match the corresponding components of the stored value.

17.5 Delegation extensions

17.5.1 Requirements

The following requirements relate to the delegation of privileges:

a) End-entity privilege certificates need to be distinguishable from AA certificates, to protect against end-entities establishing themselves as AAs without authorization. It also needs to be possible for an AA to limit the length of a subsequent delegation path.

b) An AA needs to be able to specify the appropriate name space within which the delegation of privilege can occur. Adherence to these constraints needs to be checkable by the privilege verifier.

c) An AA needs to be able to specify the acceptable certificate policies that privilege asserters further down a delegation path shall use to authenticate themselves when asserting a privilege delegation by this AA.

d) A privilege verifier needs to be able to locate the corresponding attribute certificate for an issuer to ensure that the issuer had sufficient privilege to delegate the privilege in the current certificate.

e) There is a requirement for an independent Delegation Service (DS) to issue certificates that delegate privileges, whilst the DS server cannot itself act as a claimant for those privileges.

f) An independent Delegation Service may wish to insert the name of the authority that requested the privilege assertion to be issued.

17.5.2 Delegation extensions

The following extensions are defined:

a) basic attribute constraints;

b) delegated name constraints;

c) acceptable certificate policies;

d) authority attribute identifier;

e) indirect issuer;

f) issued on behalf of; and

g) no assertion.

17.5.2.1 Basic attribute constraints extension

17.5.2.1.1 Basic attribute constraints extension definition

This extension indicates whether the subsequent delegation of the privileges assigned in the certificate containing this extension is permitted. If so, a delegation path length constraint may also be specified.

This extension is defined as follows:

```
basicAttConstraints EXTENSION ::= {
  SYNTAX          BasicAttConstraintsSyntax
  IDENTIFIED BY   {id-ce-basicAttConstraints} }

BasicAttConstraintsSyntax ::= SEQUENCE {
  authority           BOOLEAN DEFAULT FALSE,
  pathLenConstraint   INTEGER(0..MAX) OPTIONAL,
  ... }
```

The **`authority`** component indicates whether or not the holder is authorized to further delegate privilege. If **`authority`** is **`TRUE`** the holder is also an AA and is authorized to further delegate privilege, dependent on relevant constraints. If **`authority`** is **`FALSE`**, the holder is an end entity and is not authorized to delegate the privilege.

The **`pathLenConstraint`** component is meaningful only if **`authority`** is set to **`TRUE`**. It gives the maximum number of AA certificates that may follow this certificate in a delegation path. Value **`0`** indicates that the holder/subject of this certificate may issue certificates only to end entities and not to AAs. If no **`pathLenConstraint`** component appears in any certificate of a delegation path, there is no limit to the allowed length of the delegation path. Note that the constraint takes effect beginning with the next certificate in the path. The constraint controls the number of AA certificates between the AA certificate containing the constraint and end-entity certificate. The constraint restricts the length of the segment of the delegation path between the certificate containing this extension and the end-entity certificate. It has no impact on the number of AA certificates in the delegation path between the SOA/trust anchor and the certificate containing this extension. Therefore, the length of a complete delegation path may exceed the maximum length of the segment constrained by this extension. The constraint controls the number of AA certificates between the AA certificate containing the constraint and the end-entity certificate. Therefore, the total length of this segment of the path may exceed the value of the constraint by as many as two certificates. (This includes the certificates at the two endpoints of the segment plus the AA certificates between the two endpoints that are constrained by the value of this extension.)

This extension may be present in attribute certificates or public-key certificates issued by AAs, including SOAs, to other AAs or to end entities. This extension shall not be included in certificates that contain the SOA identifier extension.

If this extension is present in an attribute certificate, and **`authority`** is **`TRUE`**, the holder is authorized to issue subsequent attribute certificates delegating the contained privileges to other entities, but not public-key certificates.

If this extension is present in a public-key certificate, and if the **`basicConstraints`** extension indicates that the subject is also a CA, the subject is authorized to issue subsequent public-key certificates that delegate these privileges to other entities, but not attribute certificates. If a path length constraint is included, the subject may only delegate within the intersection of the constraint specified in this extension and any specified in the **`basicConstraints`** extension. If this extension is present in a public-key certificate but the **`basicConstraints`** extension is absent, or indicates that the subject is an end entity, the subject is not authorized to delegate the privileges.

This extension may, at the option of the certificate issuer, be either flagged as critical or as non-critical. It is recommended that it be flagged critical, otherwise a holder that is not authorized to be an AA may issue certificates and the privilege verifier may unwittingly use such a certificate.

If this extension is present and is flagged as critical, then:

– if the value of **`authority`** is not set to **`TRUE`**, then the delegated attribute shall not be used to further delegate;

– if the value of **`authority`** is set to **`TRUE`** and **`pathLenConstraint`** is present, then the privilege verifier shall check that the delegation path being processed is consistent with the value of **`pathLenConstraint`**.

If this extension is present, flagged as non-critical, and is not recognized by the privilege verifier, then that entity should use other means to determine if the delegated attributes may be used to further delegate.

If this extension is not present, or if the extension is present with an empty **`SEQUENCE`** value, the holder is constrained to being only an end entity and not an attribute authority and no delegation of the privileges contained in the attribute certificate is permitted by the holder.

17.5.2.1.2 Basic attribute constraints matching rule

The basic attribute constraints matching rule compares for equality a presented value with an attribute value of type **`AttributeCertificate`**.

```
basicAttConstraintsMatch MATCHING-RULE ::= {
  SYNTAX  BasicAttConstraintsSyntax
  ID      id-mr-basicAttConstraintsMatch }
```

This matching rule returns TRUE if the stored value contains the **`basicAttConstraints`** extension and if components that are present in the presented value match the corresponding components of the stored value.

17.5.2.2 Delegated name constraints extension

17.5.2.2.1 Delegated name constraints extension definition

The delegated name constraints extension indicates a name space within which all holder names in subsequent certificates in a delegation path need to be located.

This extension is defined as follows:

```
delegatedNameConstraints EXTENSION ::= {
  SYNTAX         NameConstraintsSyntax
  IDENTIFIED BY  id-ce-delegatedNameConstraints }
```

This extension is processed in the same manner as the **`nameConstraints`** extension for public-key certificates. If **`permittedSubtrees`** is present, of all the attribute certificates issued by the holder AA and subsequent AAs in the delegation path, only those attribute certificates with holder names within these subtrees are acceptable. If **`excludedSubtrees`** is present, any attribute certificate issued by the holder AA or subsequent AAs in the delegation path that has a holder name within these subtrees is unacceptable. If both **`permittedSubtrees`** and **`excludedSubtrees`** are present and the name spaces overlap, the exclusion statement takes precedence.

This extension may be present in attribute certificates or public-key certificates issued by AAs, including SOAs, to other AAs. This extension shall not be included in certificates issued to end-entities or certificates that contain the SOA identifier extension.

If this extension is present in a public-key certificate, and if the **`nameConstraints`** extension is also present, the subject may only delegate within the intersection of the constraint specified in this extension and that specified in the **`nameConstraints`** extension.

This extension may, at the option of the attribute certificate issuer, be either flagged as critical or as non-critical. It is recommended that it be flagged as critical, otherwise an attribute relying party may not check that subsequent attribute certificates in a delegation path are located in the name space intended by the issuing AA.

17.5.2.2.2 Delegated name constraints matching rule

The delegated name constraints matching rule compares for equality a presented value with an attribute value of type **`AttributeCertificate`**.

```
delegatedNameConstraintsMatch MATCHING-RULE ::= {
  SYNTAX  NameConstraintsSyntax
  ID      id-mr-delegatedNameConstraintsMatch }
```

This matching rule returns TRUE if the stored value contains the **attributeNameConstraints** extension and if components that are present in the presented value match the corresponding components of the stored value.

17.5.2.3 Acceptable certificate policies extension

17.5.2.3.1 Acceptable certificate policies extension definition

The acceptable certificate policies extension is used, in delegation with attribute certificates, to control the acceptable certificate policies under which the public-key certificates for subsequent holders in a delegation path need to have been issued. By enumerating a set of policies in this extension, an AA is requiring that subsequent issuers in a delegation path only delegate the contained privileges to holders that have public-key certificates issued under one or more of the enumerated certificate policies. The policies listed here are not policies under which the attribute certificate was issued, but policies under which acceptable public-key certificates for subsequent holders need to have been issued.

This extension is defined as follows:

```
acceptableCertPolicies EXTENSION ::= {
  SYNTAX         AcceptableCertPoliciesSyntax
  IDENTIFIED BY  id-ce-acceptableCertPolicies }

AcceptableCertPoliciesSyntax ::= SEQUENCE SIZE (1..MAX) OF CertPolicyId

CertPolicyId ::= OBJECT IDENTIFIER
```

This extension may only be present in attribute certificates issued by AAs, including SOAs, to other AAs. This extension shall not be included in end-entity attribute certificates or in any public-key certificates. In the case of delegation using public-key certificates, this same functionality is provided by the **certificatePolicies** and other related extensions.

If present, this extension shall be flagged as critical.

If this extension is present and the privilege verifier understands it, the verifier shall ensure that all subsequent privilege asserters in the delegation path are authenticated with a public-key certificate under one or more of the enumerated certificate policies.

If this extension is present, but not understood by the privilege verifier, the attribute certificate shall be considered invalid.

17.5.2.3.2 Acceptable certificate policies matching rule

The acceptable certificate policies matching rule compares for equality a presented value with an attribute value of type **AttributeCertificate.**

```
acceptableCertPoliciesMatch MATCHING-RULE ::= {
  SYNTAX  AcceptableCertPoliciesSyntax
  ID      id-mr-acceptableCertPoliciesMatch }
```

This matching rule returns TRUE if the stored value contains the **acceptableCertPolicies** extension and if components that are present in the presented value match the corresponding components of the stored value.

17.5.2.4 Authority attribute identifier extension

17.5.2.4.1 Authority attribute identifier extension definition

In privilege delegation, an AA that delegates privileges shall itself have at least the same privilege and the authority to delegate that privilege. An AA that is delegating privilege to another AA or to an end entity may place this extension in the AA or end-entity certificate that it issues. The extension is a back pointer to the certificate in which the issuer of the certificate containing the extension was assigned its corresponding privilege. The extension can be used by a privilege verifier to ensure that the issuing AA had sufficient privilege to be able to delegate to the holder of the certificate containing this extension.

This extension is defined as follows:

```
authorityAttributeIdentifier EXTENSION ::= {
  SYNTAX         AuthorityAttributeIdentifierSyntax
  IDENTIFIED BY  {id-ce-authorityAttributeIdentifier} }

AuthorityAttributeIdentifierSyntax ::= SEQUENCE SIZE (1..MAX) OF AuthAttId
```

```
AuthAttId ::= IssuerSerial
```

A certificate that contains this extension may include the delegation of multiple privileges to the certificate holder. If the assignment of those privileges to the AA that issued this certificate was done in more than one certificate, then this extension would include more than one pointer.

This extension may be present in attribute certificates or public-key certificates issued by AAs to other AAs or to end-entity privilege holders. This extension shall not be included in certificates issued by an SOA or in public-key certificates that contain the SOA identifier extension.

This extension shall always be flagged as non-critical.

17.5.2.4.2 AA identifier matching rule

The authority attribute identifier matching rule compares for equality a presented value with an attribute value of type **`AttributeCertificate`**.

```
authAttIdMatch MATCHING-RULE ::= {
  SYNTAX  AuthorityAttributeIdentifierSyntax
  ID      id-mr-authAttIdMatch }
```

This matching rule returns TRUE if the stored value contains the **`authorityAttributeIdentifier`** extension and if components that are present in the presented value match the corresponding components of the stored value.

17.5.2.5 Indirect issuer extension

In some environments, privilege may be delegated indirectly. In such cases, the delegator requests that a DS server issue a certificate delegating privilege on their behalf to another entity. The indirect issuer extension is used in either an attribute certificate or a public-key certificate issued to a DS server by an SOA. Presence of this extension means that the subject AA (the DS server) is authorized by that SOA to act as a proxy and issue certificates that delegate privilege, on behalf of other delegators.

```
indirectIssuer EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY   id-ce-indirectIssuer }
```

This extension shall always be flagged as non-critical.

17.5.2.6 Issued on behalf of extension

This extension is inserted into an attribute certificate by an indirect issuer (DS server). It indicates the AA that has requested the DS server to issue the attribute certificate, and allows the delegation chain to be constructed and validated.

```
issuedOnBehalfOf EXTENSION ::= {
  SYNTAX          GeneralName
  IDENTIFIED BY   id-ce-issuedOnBehalfOf }
```

The **`GeneralName`** is the name of the AA who has asked the indirect issuer (DS server) to issue this attribute certificate.

The issuer of this attribute certificate must have been granted the privilege to issue ACs on behalf of other AAs by an SOA, through the **`IndirectIssuer`** extension in its attribute certificate.

This extension may be flagged as critical or as non-critical as necessary to ensure delegation path validation.

17.5.2.7 No assertion extension

If present, this extension indicates that the attribute certificate holder cannot assert the privileges indicated in the attributes of the attribute certificate. This extension can only be inserted into AA attribute certificates, and not into end-entity attribute certificates. If present, this extension shall always be flagged as critical.

```
noAssertion EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY   id-ce-noAssertion }
```

17.6 Recognition of authority extensions

17.6.1 Requirements

The following requirements relate to recognition of authority:

a) the local SOA may wish to specify how attributes assigned in a remote domain are mapped into roles known to relying parties in the local domain;

b) the local SOA may want to constrain which privilege attributes a remote SOA is trusted to assign to which entities;

c) the local SOA may need to be able to constrain the name forms and name spaces within which a remote SOA can assign privilege attributes to users.

17.6.2 RoA extensions

The following extensions are defined:

a) Allowed attribute assignments;

b) Attribute mappings;

c) Holder name constraints.

17.6.2.1 Allowed attribute assignments extension

This extension says which privilege attributes a remote domain SOA is trusted to issue to whom.

```
allowedAttributeAssignments EXTENSION ::= {
  SYNTAX         AllowedAttributeAssignments
  IDENTIFIED BY  id-ce-allowedAttributeAssignments }

AllowedAttributeAssignments ::= SET OF SEQUENCE {
  attributes                 [0]  SET OF CHOICE {
    attributeType              [0]  AttributeType,
    attributeTypeandValues     [1]  Attribute{{SupportedAttributes}},
    ... },
  holderDomain               [1]  GeneralName,
  ... }
```

Each allowed attribute assignment comprises a set of attribute types and/or values, together with the name space which defines the holder domain. Of the name forms available through the **GeneralName** type, only those name forms that have a well-defined hierarchical structure may be used for the holder domain. The value that is specified for the holder domain forms the superior node of a subtree within which all the holder names must fall.

All the allowed attributes specified in this extension should also be specified in the attributes component of the attribute certificate. If an attribute is specified in this extension, but it is not in the attributes component, then it is ignored (i.e., it is not trusted). If an attribute is in the attributes component, but not in this extension, then it is trusted and has no further constraints on the holders to which it can be issued (other than that which might optionally be specified in the name constraints extension).

If this extension is present, it shall be flagged as critical.

17.6.2.2 Attribute mappings extension

This extension says how the attributes in the remote, trusted domain map into attributes in the local domain.

```
attributeMappings EXTENSION ::= {
  SYNTAX         AttributeMappings
  IDENTIFIED BY  id-ce-attributeMappings }

AttributeMappings ::= SET OF CHOICE {
  typeMappings       [0]  SEQUENCE {
    local              [0]  AttributeType,
    remote             [1]  AttributeType,
    ... },
  typeValueMappings  [1]  SEQUENCE {
    local              [0]  AttributeTypeAndValue,
    remote             [1]  AttributeTypeAndValue,
    ... } }
```

An attribute mapping can be at the type or value level.

When attribute mapping is at the attribute value level, each attribute value in the remote domain is mapped into an equivalent attribute value in the local domain.

NOTE 1 – Attribute value mappings may have a many-to-many relationship.

When attribute mapping is at the attribute type level, all the values assigned in the remote domain must already be understood by, and have an equal value in, the local domain.

NOTE 2 – This attribute mapping is a one-to-one mapping.

17.6.2.3 Holder name constraints extension

This extension constrains the name forms and name spaces in which a subordinate AA or a remote SOA and its subordinate AAs can issue attribute certificates.

This extension indicates that constraints are being placed on the name forms and name spaces of all name forms in attribute certificates issued by this AA and all subsequent AAs in the attribute certificate chain. If this extension is absent from all attribute certificates in an attribute certificate chain, then no constraints are placed on any name spaces in the attribute certificate chain. If this extension is present in an attribute certificate, then constraints are automatically placed on the name spaces of every name form in the attribute certificate chain from this point onwards, regardless of whether the name form is explicitly included in the extension or not, i.e., the default constraint on each name form excludes the entire name space.

NOTE – Because there can be an unbounded set of registeredID name forms, then it is not possible for new name forms to be unconstrained once this extension is present, without the name form being explicitly included in this extension via a permitted subtree.

This extension is defined as follows:

```
holderNameConstraints EXTENSION ::= {
  SYNTAX         HolderNameConstraintsSyntax
  IDENTIFIED BY  id-ce-holderNameConstraints }

HolderNameConstraintsSyntax ::= SEQUENCE {
  permittedSubtrees   [0]  GeneralSubtrees,
  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 **permittedSubtrees** and **excludedSubtrees** components each specify one or more naming subtrees of one or more name forms. Each subtree is defined by the name of the root of the subtree, i.e., the **base** component, and, optionally, within that subtree, an area that is bounded by upper and/or lower levels.

An empty distinguished name sequence is equivalent to a wildcard and means that all DNs fall within the subtree.

The **minimum** component specifies the upper bound of the area within the subtree. All names whose final name component is above the level specified are not contained within the area. A value of **minimum** equal to zero (the default) corresponds to the base, i.e., the top node of the subtree. For example, if **minimum** is set to one, then the naming subtree excludes the base node but includes subordinate nodes.

The **maximum** component specifies the lower bound of the area within the subtree. All names whose last component is below the level specified are not contained within the area. A value of **maximum** of zero corresponds to the base, i.e., the top of the subtree. An absent **maximum** component indicates that no lower limit should be imposed on the area within the subtree. For example, if **maximum** is set to one, then the naming subtree excludes all nodes except the subtree base and its immediate subordinates.

The **permittedSubtrees** component is used to reduce the constraints placed on the name spaces of one or more name forms. Since the entire name space of each form is automatically fully excluded when this extension appears in an AA certificate, the **permittedSubtrees** component describes the name space(s) that is(are) permitted. If an entire name space of a particular name form is to be permitted, this is achieved by setting the **base** component to the root of the name space.

The optional **excludedSubtrees** component is used to exclude one or more subordinate subtrees from the **permittedSubtrees**. For example, if in the ITU-T X.500 distinguished name space, the subtree C=GB is permitted, but the subtrees C=GB, O=XYZ and C=GB, O=ABC are not permitted, then the **permittedSubtrees** will be set to C=GB and the **excludedSubtrees** will be set to C=GB, O=XYZ and C=GB, O=ABC. If the **excludedSubtrees** is present and its name spaces overlap with the **permittedSubtrees**, the **excludedSubtrees** statement takes precedence.

All holder names in subsequent attribute certificates in a delegation path shall be located in the permitted name spaces for the attribute certificates to be acceptable. When a certificate holder has multiple names of the same name form then all such names shall be located in the permitted name space of that name form for the certificate to be acceptable. When a certificate holder has multiple names in different name forms, each name shall be located in the permitted name space of that name form for the certificate to be acceptable.

Of the name forms available through the **GeneralName** type, only those name forms that have a well-defined hierarchical structure may be used in these components.

The **directoryName** name form satisfies this requirement; when using this name form, a naming subtree corresponds to a DIT subtree. An attribute certificate is considered subordinate to the **base** (and therefore a candidate to be within the subtree) if the sequence of RDNs, which forms the full distinguished name in **base**, matches the initial sequence of the same number of RDNs which forms the first part of the distinguished name of the holder of the attribute certificate. The distinguished name of the holder of the certificate may have additional trailing RDNs in its sequence that do not appear in the distinguished name in **base**. The **distinguishedNameMatch** matching rule is used to compare the value of **base** with the initial sequence of RDNs in the distinguished name of the subject of the certificate.

Conformant implementations are not required to recognize all possible name forms. If a privilege verifier does not recognize a name form used in any **base** component, and

– that name form also occurs in the **holder** component of a subsequent attribute certificate in the chain, then that attribute certificate shall be handled as if an unrecognized critical extension had been encountered; or

– that name form does not occur in the **holder** component of a subsequent attribute certificate in the chain, then this name form can be ignored.

If a privilege verifier does not recognize a name form that occurs in the **holder** component of a subsequent attribute certificate in the chain from that in which this extension appeared, but that name form does not occur in any **base** component of this extension, then that attribute certificate shall be rejected.

This extension shall always be flagged as critical.

A privilege verifier shall check that the attribute certification path being processed is within the constraints specified by the value in this extension.

17.6.2.4 Relationship of delegated name constraints to holder name constraints

The **delegatedNameConstraints** extension described in clause 17.5.2.2 has the same semantics as the **nameConstraints** extension of public-key certificates, which is that every name form is allowed unless specifically constrained. The **holderNameConstraints** extension on the other hand, whilst having the same syntax, has the opposite semantics; which is that, once the extension is present, every name form is denied unless specifically permitted. If both the **delegatedNameConstraints** extension and the **holderNameConstraints** extension appear in the same attribute certificate, then the excluded name spaces are the union of the excluded name spaces from both extensions, whilst the included name spaces are the intersection of the name spaces from both extensions.

17.7 Use of basic CRL extension for ACRLs

17.7.1 Requirements

17.7.2 Use of CRL extensions

17.7.2.1 Use of CRL number extension for ACRL

This ACRL extension conveys a monotonically increasing sequence number for each ACRL issued by a given ACRL issuer through a given ACRL directory attribute (see clauses 19.2.4 to 19.2.6) or ACRL distribution point. It allows a privilege verifier to detect whether ACRLs issued prior to the one being processed were also seen and processed. This extension syntax is as defined in clause 9.5.2.1.

This extension shall always be flagged as non-critical.

NOTE - The extension defined here is relevant for both CRLs and ACRLs). Only ACRL aspects are considered here. For ARL aspects, see clause 9.5.2.1.

17.7.2.2 Use of status referral extension for ACRL

The status referral extension is defined in 9.5.2.2 for use in CRLs. This extension may also be used in ACRLs.

The procedure specified in 9.5.2.2 also applies here by replacing public-key certificate, CRL and relying party with attribute certificate, ACRL and privilege verifier, respectively.

17.7.2.3 Use of CRL stream identifier for ACRL

The CRL stream identifier extension is defined in 9.5.2.3 for use in CRLs. This extension may also be used in ACRLs.

The procedure specified in 9.5.2.3 also applies here by replacing CRL with ACRL.

17.7.2.4 Use of order list extension for ACRL

The CRL stream identifier extension is defined in 9.5.2.4 for use in CRLs. This extension may also be used in ACRLs.

The procedure specified in 9.5.2.4 also applies here by replacing public-key certificate and CRL with attribute certificate and ACRL, respectively.

17.7.2.5 Use of delta information extension for ACRL

The delta information extension is defined in 9.5.2.5 for use in CRLs. This extension may also be used in ACRLs.

The procedure specified in 9.5.2.5 also applies here by replacing CRL, dCRL and relying party with ACRL, dACRL and privilege verifier, respectively.

17.7.2.6 Use of to be revoked extension for ACRL

This ACRL extension allows for the notification that attribute certificates will be revoked as of a specified date and time in the future. The **toBeRevoked** extension is used to specify the reason for the attribute certificate revocation, the date and time at which the attribute certificate will be revoked, and the group of attribute certificates to be revoked. Each list can contain a single attribute certificate serial number, a range of attribute certificate serial numbers or a named **subtree**.

NOTE – The following ASN.1 is copied from 9.5.2.6 for easy reference.

```
toBeRevoked EXTENSION ::= {
  SYNTAX          ToBeRevokedSyntax
  IDENTIFIED BY   id-ce-toBeRevoked }

ToBeRevokedSyntax ::= SEQUENCE SIZE (1..MAX) OF ToBeRevokedGroup

ToBeRevokedGroup ::= SEQUENCE {
  certificateIssuer   [0]   GeneralName OPTIONAL,
  reasonInfo          [1]   ReasonInfo OPTIONAL,
  revocationTime            GeneralizedTime,
  certificateGroup          CertificateGroup,
  ... }

ReasonInfo ::= SEQUENCE {
  reasonCode            CRLReason,
  holdInstructionCode   HoldInstruction OPTIONAL,
  ... }

CertificateGroup ::= CHOICE {
  serialNumbers       [0]   CertificateSerialNumbers,
  serialNumberRange   [1]   CertificateGroupNumberRange,
  nameSubtree         [2]   GeneralName,
  ... }

CertificateGroupNumberRange ::= SEQUENCE {
  startingNumber   [0]   INTEGER,
  endingNumber     [1]   INTEGER,
  ... }

CertificateSerialNumbers ::= SEQUENCE SIZE (1..MAX) OF CertificateSerialNumber
```

The **certificateIssuer** component, if present, identifies the AA that issued all the attribute certificates listed in this ToBeRevokedGroup. If certificateIssuer is omitted, it defaults to the ACRL issuer name.

The **reasonInfo** component, if present, identifies the reason for the attribute certificate revocations. If present, this extension indicates that all attribute certificates identified in **ToBeRevokedGroup** will be revoked for the reason indicated in this component. If the **reasonCode** component contains the value **certificateHold**, the **holdInstructionCode** component may also be present. If present, the **holdInstructionCode** component indicates the action to be taken on encountering any of the attribute certificates identified in **RevokedGroup**. This action should only be taken, after the revocation time indicated in the **revocationTime** component has passed.

The **revocationTime** component indicates the date and time at which this group of attribute certificates will be revoked and should therefore be considered invalid. This date shall be later than the **thisUpdate** time of the ACRL containing this extension. If **revocationTime** is before the **nextUpdate** time of the ACRL containing this extension, the attribute certificates shall be considered revoked between the **revocationTime** and the **nextUpdate** time by a relying party using an ACRL containing this extension. Otherwise, this is a notice that at a specified time in the future these attribute certificates will be revoked. Once the revocation time has passed, either the AA has revoked the attribute certificate or not. If it has revoked the attribute certificate, future ACRLs shall include this on the list of revoked certificates, at least until the attribute certificate expires. If the AA has not revoked the attribute certificate, but still intends to revoke it in the future, it may include the attribute certificate in this extension on subsequent ACRLs with a revised **revocationTime**. If the AA no longer intends to revoke the attribute certificate, it may be excluded from all subsequent ACRLs and the attribute certificate shall not be considered revoked.

The **certificateGroup** component lists the set of public-key certificates to be revoked. This component identifies the attribute certificates issued by the AA identified in the **certificateIssuer** to be revoked at the date/time identified in **revocationTime**. This set of public-key certificates is not further refined by any outside controls (e.g., **issuingDistributionPoint**).

The **serialNumbers** component, if present, shall hold the serial number(s) of the attribute certificate(s) issued by the identified issuing AA that will be revoked at the specified time.

If the **serialNumberRange** component is present, all attribute certificates in the range beginning with the starting serial number and ending with the ending serial number and issued by the identified issuing AA will be revoked at the specified time.

If the **nameSubtree** component is present, all attribute certificates with a holder name that is subordinate to the specified name and issued by the identified issuing AA will be revoked at the specified time. If the **nameSubtree** contains a distinguished name then all distinguished names associated with the subject of an attribute certificate (i.e., holder component of an attribute certificate) need to be considered. For other name forms, the holder component of attribute certificates need to be considered. If at least one of the names associated with the holder, contained in the attribute certificate, is within the subtree specified in **nameSubtree**, that attribute certificate will be revoked at the specified time. As with the **nameConstraints** extension, not all name forms are appropriate for **subtree** specification. Only those that have recognized subordination rules should be used in this extension.

This extension may, at the option of the ACRL issuer, be flagged as critical or as non-critical. As the information provided in this extension applies to revocations that will occur in the future, it is recommended that it be flagged as non-critical, reducing the risk of problems with interoperability and backward compatibility.

17.7.2.7 Use of revoked group of certificates extension

A set of attribute certificates that have been revoked can be published using the following CRL extension. Each list of attribute certificates to be revoked is associated with a specific attribute certificate issuer and revocation time. Each list can contain a range of attribute certificate serial numbers or a named subtree.

NOTE – The following ASN.1 is copied from 9.5.2.7 for easy reference.

```
revokedGroups EXTENSION ::= {
  SYNTAX          RevokedGroupsSyntax
  IDENTIFIED BY   id-ce-revokedGroups }

RevokedGroupsSyntax ::= SEQUENCE SIZE (1..MAX) OF RevokedGroup

RevokedGroup ::= SEQUENCE {
  certificateIssuer         [0]  GeneralName OPTIONAL,
  reasonInfo                [1]  ReasonInfo OPTIONAL,
  invalidityDate            [2]  GeneralizedTime OPTIONAL,
  revokedcertificateGroup   [3]  RevokedCertificateGroup,
  ... }

RevokedCertificateGroup ::= CHOICE {
  serialNumberRange  NumberRange,
  nameSubtree        GeneralName }
```

The **certificateIssuer** component, if present, identifies the AA that issued all the attribute certificates listed in this RevokedGroup. If certificateIssuer is omitted, it defaults to the ACRL issuer name.

The **reasonInfo** component, if present, identifies the reason for the attribute certificate revocations. If present, this component indicates that all attribute certificates identified in **RevokedGroup** component were revoked for the reason indicated in this component. If the **reasonCode** contains the value **certificateHold**, the **holdInstructionCode** may also be present. If present, **`holdInstructionCode`** indicates the action to be taken on encountering any of the attribute certificates identified in the value of the **`RevokedGroup`** data type.

The **`invalidityDate`** component, if present, indicates the time from which all attribute certificates identified in **`RevokedGroup`** should be considered invalid. This date shall be earlier than the date contained in **`thisUpdate`** component of the ACRL. If omitted, all attribute certificates identified in **`RevokedGroup`** should be considered invalid at least from the time indicated in the **`thisUpdate`** component of the ACRL. If the status of the attribute certificate prior to the **`thisUpdate`** time is critical to a privilege verifier (e.g., to determine whether a digital signature that was created prior to this ACRL issuance occurred while the attribute certificate was still valid or after it had been revoked), additional revocation status checking techniques will be required to determine the actual date/time from which a given attribute certificate should be considered invalid.

The **`revokedCertificateGroup`** component lists the set of attribute certificates that have been revoked. This component identifies the attribute certificates issued by the AA identified in **`certificateIssuer`** revoked under the specified conditions. This set of attribute certificates is not further refined by any outside controls (e.g., **`issuingDistributionPoint`**).

If the **`serialNumberRange`** is present, all attribute certificates containing certificate serial numbers within the specified range, issued by the identified AA are applicable.

If the **`nameSubtree`** component is present, all certificates with a holder name that is subordinate to the specified name and issued by the identified AA will be revoked at the specified time. If the **`nameSubtree`** contains a distinguished name then all distinguished names associated with the **`holder`** component of an attribute certificate need to be considered. For other name forms, the holder component of attribute certificates needs to be considered. If at least one of the names associated with the holder, contained in the certificate, is within the subtree specified in **`nameSubtree`**, that attribute certificate has been revoked. As with the **`nameConstraints`** extension, not all name forms are appropriate for subtree specification. Only those that have recognized subordination rules should be used in this extension.

This extension is always flagged as critical. Otherwise, a privilege verifier may incorrectly assume that attribute certificates, identified as revoked within this extension, are not revoked. When this extension is present it may be the only indication of revoked attribute certificates in an ACRL (i.e., the **`revokedCertificates`** component of the ACRL may be empty) or it may list revoked attribute certificates that are in addition to those indicated in the **`revokedCertificates`** component. A revoked attribute certificate shall not be listed both in the **`revokedCertificates`** component and in this extension.

17.7.2.8 Use of expired certificates on ACRL extension

The use of expired certificates extension is defined in 9.5.2.8 for use in CRLs. This extension may also be used in ACRLs.

The procedure specified in 9.5.2.8 also applies here by replacing public-key certificate and CRL with attribute certificate and ACRL, respectively.

17.7.3 Use of CRL entry extensions

17.7.3.1 Use of reason code extension

The reason code extension is specified in clause 9.5.3.1 for use in CRL entries. This extension may also be used in ACRL entries

The meaning of the reason codes defined in clause 9.5.3.1 as they relate to ACRL entries are as follows:

– **`unspecified`** can be used to revoke attribute certificates for reasons other than the specific codes.

– **`keyCompromise`** is not relevant for ACRLs.

– **`cACompromise`** is not relevant for ACRLs.

– **`affiliationChanged`** indicates that information in the attribute certificate has been modified.

– **`superseded`** indicates that the attribute certificate has been superseded.

– **`cessationOfOperation`** indicates that the attribute certificate is no longer needed for the purpose for which it was issued.

– **`privilegeWithdrawn`** indicates that the attribute certificate was revoked because a privilege contained within that attribute certificate has been withdrawn.

– **`aACompromise`** indicates that it is known or suspected that aspects of the AA validated in the attribute certificate have been compromised.

The procedure specified in clause 9.5.3.1 also applies here by substituting public-key certificate and CRL entry with attribute certificate and ACRL entry, respectively.

17.7.3.2 Use of hold instruction code extension

The use of hold instruction code extension is defined in 9.5.3.2 for use in CRL entries. This extension may also be used in ACRL entries.

The procedure specified in 9.5.3.2 also applies here by replacing public-key certificate and CRL with attribute certificate and ACRL, respectively.

18 Delegation path processing procedure

Delegation path processing is carried out by a privilege verifier. The path processing rules for attribute certificates are somewhat analogous to those for public-key certificates.

Other components of the path processing that are not addressed in this clause include verification of certificate signatures, validation of certificate validity periods, etc.

For delegation paths consisting of a single certificate (i.e., the privileges were assigned directly to the privilege asserter by the SOA), only the basic procedure, as described in clause 18.1 below is required, unless the privilege is assigned to a role. In that case, if the privilege verifier is not configured with the specific privileges of the role, it may need to obtain the role specification certificate that assigns the specific privileges to the role as described in clause 18.2 below. If the privilege asserter was delegated its privilege by an intermediary AA, then the delegation path procedure in clause 18.3 is also required. These procedures are not performed sequentially. The role processing procedure and delegation processing procedure are done prior to the determination of whether or not the asserted privileges are sufficient for the context of use within the basic procedure.

18.1 Basic processing procedure

The signature on every certificate in the path shall be verified. Procedures related to validating signatures and public-key certificates are not repeated in this clause. The privilege verifier shall verify the identity of every entity in the path, using the procedures of clause 12. Note that checking the signature on an attribute certificate necessarily involves checking the referenced public-key certificate for its validity. Where privileges are assigned using attribute certificates, path processing engines will need to consider elements of both the PMI and the PKI in the course of determining the ultimate validity of a privilege asserter's attribute certificate. Not all attribute certificate issuers need have public-key certificates issued by the same trust anchor CA (or one of its subordinate CAs), in which case multiple PKI certification paths will need to be followed. Once that validity has been confirmed, the privileges contained in that certificate *may* be used depending on a comparison with the relevant privilege policy and other information associated with the context in which the certificate is being used.

The context of use shall determine if the privilege holder actually intended to assert the contained privilege for use with that context. The fact that a chain of certificates to a trusted SOA exists is not in itself enough upon which to make this determination. The willingness of the privilege holder to use that certificate has to be clearly indicated and verified. However, mechanisms to ensure that such a privilege assertion has been adequately demonstrated by the privilege holder are outside the scope of this Specification. As an example, such a privilege assertion may be verifiable if the privilege holder signed a reference to that certificate, thereby indicating their willingness to use that certificate for that context.

For each attribute certificate in the path that does not contain the **`noRevAvail`** extension, the privilege verifier shall ensure that the attribute certificate has not been revoked.

The privilege verifier shall ensure that the asserted privilege is valid for the time called "time of evaluation" which can be done for *any* time, i.e., the current time of checking or any time in the past. In the context of an access control service, the checking is always done for the present time. However, in the context of non-repudiation, the checking can be done for a time in the past or the current time. When certificates are validated, the privilege verifier shall ensure that the time of evaluation falls within all the validity periods of all the certificates used in the path. Also, if any of the certificates in the path contain the **`timeSpecification`** extension, the constraints placed over the times the privilege can be asserted need to also allow the privilege assertion to be valid at the time of evaluation.

If the **`targetingInformation`** extension is present in the certificate used to assert a privilege, the privilege verifier shall check that the server/service for which it is verifying is included in the list of targets.

If the **`singleUse`** extension that is present in the attribute certificate is used to assert a privilege, the privilege verifier shall check that the attribute certificate has not been asserted prior to the current use.

If the attribute certificate is a role assignment certificate, the processing procedure described in clause 18.2 is needed to ensure that the appropriate privileges are identified. If the privilege was delegated to the entity rather than assigned directly by the SOA trusted by the privilege verifier, the processing procedure described in clause 18.3 is needed to ensure that delegation was done properly.

The privilege verifier shall also determine whether or not the privileges being asserted are sufficient for the context of use. The privilege policy establishes the rules for making this determination and includes the specification of any environmental variables that need to be considered. The privileges asserted, including those resulting from the role procedure in clause 18.2 and the delegation procedure in clause 18.3 and any relevant environmental variables (e.g., time of day or current account balance) are compared against the privilege policy to determine whether or not they are sufficient for the context of use. If the **`acceptablePrivilegePolicies`** extension is present, the privilege assertion can only succeed if the privilege policy the privilege verifier is comparing against is one of those contained in this extension.

If the comparison succeeds, any relevant user notices are provided to the privilege verifier.

18.2 Role processing procedure

If the asserted certificate is a role assignment certificate, the privilege verifier shall obtain the specific privileges assigned to that role. The name of the role to which the privilege asserter is assigned is contained in the **`role`** attribute of the certificate. The privilege verifier, if not already configured with the privileges of the named role, may need to locate the role specification certificate that assigns the privileges to that role. Information in the **`role`** attribute and in the **`roleSpecCertIdentifier`** extension may be used to locate that certificate.

The privileges assigned to the role are implicitly assigned to the privilege asserter and are therefore included among the asserted privileges that are compared against the privilege policy in the basic procedure in clause 18.1 to determine whether or not the asserted privileges are sufficient for the context of use.

18.3 Delegation processing procedure

If the privileges asserted are delegated to the privilege asserter by an intermediary AA, the privilege verifier shall ensure that the path is a valid delegation path, by ensuring that:

- each AA that issued a certificate in the delegation path was authorized to do so;
- each certificate in the delegation path is valid with respect to path and name constraints imposed on it;
- each entity in the delegation path is authenticated with a public-key certificate that is valid according to any imposed policy constraints;
- no AA delegation privilege is greater than the privilege held by that AA.

In complex delegation-of-authority scenarios, where the delegations form a directed graph, with multiple trusted root SOAs, it is possible for an AA to combine the privilege attributes it holds in two or more ACs and to delegate a combination of these attributes to a subordinate in a single, delegated attribute certificate. Validating these split delegation paths in directed graphs is much more complex than validating a simple path through a hierarchical tree of ACs that lead from a single root SOA. Implementations need to consider carefully whether to allow directed graph type delegations or to limit delegations to a simple tree structure.

Prior to commencing delegation path validation, the privilege verifier shall obtain the following. Any of these may be provided by the privilege asserter, or obtained by the privilege verifier from another source, such as the Directory. The attributes of the service may be provided to the privilege verifier in a structured document or by other means.

- Established trust in the public verification key used to validate the trusted SOA's signature. This trust can either be established through out-of-band means or through a public-key certificate issued to the SOA by a CA in which the privilege verifier already has established trust. Such a certificate would contain the **`sOAIdentifier`** extension.
- The privilege asserter's privilege, encoded in their attribute certificate or subject directory attributes extension of their public-key certificate.
- Delegation path of certificates from the privilege asserter to the trusted SOA.
- Domination rule for the privilege being asserted; this may be obtained from the attribute descriptor issued by the SOA responsible for the attribute in question or it may be obtained through out-of-band means.
- Privilege policy; this may be obtained from a directory or from some out-of-band means.
- Environmental variables, including for example, current date/time, current account balance, etc.

An implementation shall be functionally equivalent to the external behaviour resulting from this procedure; however, the algorithm used by a particular implementation to derive the correct output(s) from the given inputs is not standardized.

In the case where attribute certificates are issued by an indirect issuer (DS), which does not have a full set of privileges directly assigned to it, the privilege verifier should fully validate the delegation chain as follows:

i) Starting with the end-entity attribute certificate, the privilege verifier extracts the issuer name and the **`issuedOnBehalfOf`** name.

ii) The privilege verifier retrieves the attribute certificate of the issuer and validates that the issuer is an indirect issuer of the SOA (i.e., has the **`indirectIssuer`** extension).

iii) The privilege verifier retrieves the attribute certificate of the **`issuedOnBehalfOf`** AA and validates that the AA has a superset of the privilege attributes issued to the end entity.

However, in order to aid path determination and validation, certificates may contain the authority information access and authority key identifier extensions, whose usage is described in clause 18.3.1 below.

The privilege verifier recurses to step ii) using the attribute certificate of the AA, and thereby moves up the chain until it arrives at the attribute certificate of an AA that is issued by the SOA.

18.3.1 Verify integrity of domination rule

The domination rule is associated with the privilege being delegated. The syntax and method for obtaining the domination rule is not standardized. However, the integrity of the retrieved domination rule can be verified. The attribute descriptor certificate issued by the SOA responsible for the attribute being delegated may contain a HASH of the domination rule. The privilege verifier may reproduce the HASH function on the retrieved copy of the domination rule and compare the two hashes. If they are identical, the privilege verifier has the accurate domination rule.

18.3.2 Establish valid delegation path

The privilege verifier shall find the delegation path and obtain certificates for every entity in the path. The delegation path extends from the direct privilege asserter to the SOA. Each intermediary certificate in the delegation path shall contain the **`basicAttConstraints`** extension with the authority component set to **`TRUE`**. The issuer of each certificate shall be the same as the holder/subject of the certificate which is adjacent to it in the delegation path. The **`authorityAttributeIdentifier`** extension is used to identify the certificate(s) of the issuer of the current certificate in the delegation path. The **`authorityInformationAccess`** extension may be used to locate the appropriate certificates of the issuer of the current certificate in the delegation path, as described in clause 18.3.2.1 below. The **`authorityKeyIdentifier`** extension may be used to locate and identify the public key of the issuer of the current certificate in the delegation path, as described in clause 18.3.2.2 below. The number of certificates in the path from each entity to the direct privilege asserter (inclusive) shall not exceed the value of the **`pathLenConstraint`** value in the entity's **`basicAttConstraints`** extension by more than 2. This is because the **`pathLenConstraint`** limits the number of intermediary certificates between the two endpoints (i.e., the certificate containing the constraint and the end-entity certificate) so the maximum length is the value of that constraint plus the certificates that are the endpoints.

If **`delegatedNameConstraints`** extension is present in any of the certificates in the delegation path, the constraints are processed in the same way as the **`nameConstraints`** extension is processed in the certification path processing procedure in clause 12.

If the **`acceptableCertPolicies`** extension is present in any of the certificates in the delegation path, the privilege verifier shall ensure that the authentication of each subsequent entity in the delegation path is done with a public-key certificate that contains at least one of the acceptable policies.

18.3.2.1 Use of authority information access extension

The authority information access (AIA) extension is defined in IETF RFC 5280.

The AIA extension indicates how to access information and services for the issuer of the certificate in which the extension appears. In the context of attribute certificates, it is used to point to information about the AA that issued the attribute certificate in which it appears. This information may include online validation services and AA policy data. (Note that the location of ACRLs is not specified in this extension.) This extension may be included in end-entity or AA ACs, and it MUST be non-critical.

Each entry in the sequence **`AuthorityInfoAccessSyntax`** describes the format and location of additional information provided by the AA that issued the attribute certificate in which this extension appears. The type and format of the additional information is specified by the **`AccessMethod`** component; the **`accessLocation`** component specifies the location of this additional information. The retrieval mechanism may be implied by the **`accessMethod`** or specified by **`accessLocation`**.

In an attribute certificate, the **`id-ad-caIssuers`** OID is used when the additional information lists ACs that were issued to and used by the AA to issue the attribute certificate containing this extension. The referenced attribute certificate(s) is/are intended to aid relying parties in the selection of an attribute certification path that terminates at a point (SOA or AA) trusted by the relying party.

When the **`id-ad-caIssuers`** OID appears as an **`accessMethod component`**, the **`accessLocation`** component describes the referenced description server and the access protocol to obtain the referenced ACs. The **`accessLocation`** component is defined as a **`GeneralName`**, which can take several forms. Where the information is available via HTTP, FTP, or LDAP, **`accessLocation`** should be a **`uniformResourceIdentifier`**.

The LDAP URI should specify a **`distinguishedName`** and an attribute and may specify a host name, for example:

ldap://ldap.example.com/cn=Some%20Manager,dc=example,dc=com?attributeCertificateAttribute;binary

Omitting the host name (e.g., ldap:///cn=Some%20Manager,dc=example,dc=com?attributeCertificateAttribute;binary) has the effect of specifying the use of whatever LDAP server is locally configured. The URI should list the appropriate attribute description for the attribute holding DER encoded ACs. Note that in LDAP it is generally not possible to specify the exact set of ACs that were used to issue the attribute certificate containing this extension, but rather the **`accessLocation`** points to all the ACs belonging to the issuer of the attribute certificate containing this extension.

The ftp and http URIs should specify either the single DER encoded attribute certificate that was used to issue the attribute certificate containing this extension, or a filestore directory containing the set of ACs belonging to the issuer of the attribute certificate containing this extension. Individual DER encoded attribute certificates should have a file name ending in .ace, for example:

http://www.example.com/ACs/dc=com/dc=example/cn=Some%20Manager/leader.ace

The filestore directory containing the complete set of ACs for the same entity might be:

ftp://www.example.com/ACs/dc=com/dc=example/cn=Some%20Manager/

Where the information is available via the Directory Access Protocol (DAP), **`accessLocation`** should be a **`directoryName`**. The entry for that **`directoryName`** contains AA ACs in the **`attributeCertificateAttribute`** attribute. When the information is available via electronic mail **`accessLocation`** should be an **`rfc822Name`**. The semantics of other **`caIssuers accessLocation`** name forms are not defined.

18.3.2.2 Use of authority key identifier

The AKI is used to identify the public key to be used to verify the signature on the attribute certificate in which this extension occurs. It is recommended that the **`authorityCertIssuer`** component and the **`authorityCertSerialNumber`** component are used together to identify and optionally locate the public-key certificate of the attribute certificate issuer as follows. The **`GeneralNames`** of the **`authorityCertIssuer`** component should be used to name the CA which issued the public-key certificate and also to optionally identify where the public-key certificate can be found when it is available via http, ftp, or ldap. In the latter case, one of the **`GeneralNames`** should be a **`uniformResourceIdentifier`** as specified in clause 18.3.2.1, and should point to either the LDAP entry holding the public key-certificate or the filestore directory holding the public-key certificate or the actual file containing the public-key certificate of the attribute certificate issuer. The **`authorityCertSerialNumber`** component is used to specify the serial number of the specific public-key certificate to be used, from the possible set of public-key certificates issued to the attribute certificate issuer.

18.3.3 Verify privilege delegation

No delegator can delegate privilege that is greater than the privilege they own. The domination rule in the attribute descriptor attribute provides the rules for when a given value is 'less than' another value for the attribute being delegated.

For each certificate in the delegation path, including the direct privilege asserter's certificate, the privilege verifier shall ensure that the delegator was authorized to delegate the privilege they own and that the privilege delegated was not greater than the privilege owned.

For each of these certificates, the privilege verifier shall compare the delegated privilege with the privilege owned by that delegator, in accordance with the domination rule for the privilege. The privilege owned by the delegator is obtained from the adjacent certificate in the delegation path, as described in clause 18.2. The comparison of the two privileges is done based on the domination rule discussed in clause 18.3.1.

18.3.4 Pass/fail determination

Assuming that a valid delegation path is established, the privileges of the direct privilege asserter are provided as input for the comparison against the privilege policy as discussed in clause 18.1 to determine whether or not the direct privilege asserter has sufficient privilege for the context of use.

19 PMI directory schema

This clause defines the directory schema elements used to represent PMI information in the Directory. It includes specification of relevant object classes, attributes and attribute value matching rules.

19.1 PMI directory object classes

This subclause defines object class definitions for representing PMI objects in the Directory.

19.1.1 PMI user object class

The PMI user object class is used in defining entries for objects that may be the holder of attribute certificates.

```
pmiUser OBJECT-CLASS ::= {
  SUBCLASS OF   {top}
  KIND          auxiliary
  MAY CONTAIN   {attributeCertificateAttribute}
  ID            id-oc-pmiUser }
```

19.1.2 PMI AA object class

The PMI AA object class is used in defining entries for objects that act as attribute authorities.

```
pmiAA OBJECT-CLASS ::= { -- a PMI AA
  SUBCLASS OF   {top}
  KIND          auxiliary
  MAY CONTAIN   {aACertificate |
                 attributeCertificateRevocationList |
                 eeAttrCertificateRevocationList |
                 attributeAuthorityRevocationList}
  ID            id-oc-pmiAA }
```

19.1.3 PMI SOA object class

The PMI SOA object class is used in defining entries for objects that act as sources of authority. Note that if the object was authorized to act as an SOA through issuance of a public-key certificate containing the **sOAIdentifier** extension, a directory entry representing that object would also contain the **pkiCA** object class.

```
pmiSOA OBJECT-CLASS ::= { -- a PMI Source of Authority
  SUBCLASS OF   {top}
  KIND          auxiliary
  MAY CONTAIN   {attributeCertificateRevocationList |
                 eeAttrCertificateRevocationList |
                 attributeAuthorityRevocationList |
                 attributeDescriptorCertificate}
  ID            id-oc-pmiSOA }
```

19.1.4 Attribute certificate CRL distribution point object class

The attribute certificate CRL distribution point object class is used in defining entries for objects that contain attribute certificate and/or attribute authority revocation list segments. This auxiliary class is intended to be combined with the **crlDistributionPoint** structural object class when instantiating entries. Since the **certificateRevocationList** and **authorityRevocationList** attributes are optional in that class, it is possible to create entries which contain, for example, only an attribute authority revocation list or entries which contain revocation lists of multiple types, depending on the requirements.

```
attCertCRLDistributionPt OBJECT-CLASS ::= {
  SUBCLASS OF   {top}
  KIND          auxiliary
  MAY CONTAIN   {attributeCertificateRevocationList |
                 eeAttrCertificateRevocationList |
                 attributeAuthorityRevocationList}
  ID            id-oc-attCertCRLDistributionPts }
```

19.1.5 PMI delegation path object class

The PMI delegation path object class is used in defining entries for objects that may contain delegation paths. It will generally be used in conjunction with entries of structural object class **pmiAA**.

```
pmiDelegationPath OBJECT-CLASS ::= {
  SUBCLASS OF   {top}
  KIND          auxiliary
  MAY CONTAIN   {delegationPath}
  ID            id-oc-pmiDelegationPath }
```

19.1.6 Privilege policy object class

The privilege policy object class is used in defining entries for objects that contain privilege policy information.

```
privilegePolicy OBJECT-CLASS ::= {
  SUBCLASS OF   {top}
  KIND          auxiliary
  MAY CONTAIN   {privPolicy}
  ID            id-oc-privilegePolicy }
```

19.1.7 Protected privilege policy object class

The protected privilege policy object class is used in defining entries for objects that contain privilege policies protected within attribute certificates.

```
protectedPrivilegePolicy OBJECT-CLASS ::= {
  SUBCLASS OF   {top}
  KIND          auxiliary
  MAY CONTAIN   {protPrivPolicy}
  ID            id-oc-protectedPrivilegePolicy }
```

19.2 PMI directory attributes

This subclause defines directory attributes used to store PMI data in directory entries.

19.2.1 Attribute certificate attribute

The following attribute contains end-entity attribute certificates issued to a specific holder and is stored in the directory entry of that holder.

```
attributeCertificateAttribute ATTRIBUTE ::= {
  WITH SYNTAX             AttributeCertificate
  EQUALITY MATCHING RULE  attributeCertificateExactMatch
  ID                      id-at-attributeCertificate }
```

19.2.2 AA certificate attribute

The following attribute contains attribute certificates issued to an AA and is stored in the directory entry of the holder AA.

```
aACertificate ATTRIBUTE ::= {
  WITH SYNTAX             AttributeCertificate
  EQUALITY MATCHING RULE  attributeCertificateExactMatch
  ID                      id-at-aACertificate }
```

19.2.3 Attribute descriptor certificate attribute

The following attribute contains attribute certificates issued by an SOA that contain the **attributeDescriptor** extension. These attribute certificates contain the valid syntax and domination rule specification of privilege attributes and is stored in the directory entry of the issuing SOA.

```
attributeDescriptorCertificate ATTRIBUTE ::= {
  WITH SYNTAX             AttributeCertificate
  EQUALITY MATCHING RULE  attributeCertificateExactMatch
  ID                      id-at-attributeDescriptorCertificate }
```

19.2.4 Attribute certificate revocation list attribute

The following attribute contains a list of revoked attribute certificates. These lists may be stored in the directory entry of the issuing authority, or other directory entry (e.g., a distribution point).

```
attributeCertificateRevocationList ATTRIBUTE ::= {
  WITH SYNTAX             CertificateList
  EQUALITY MATCHING RULE  certificateListExactMatch
  LDAP-SYNTAX             x509CertificateList.&id
  LDAP-NAME               {"AttrCertificateRevocationList"}
  LDAP-DESC               "X.509 Attr certificate revocation list"
  ID                      id-at-attributeCertificateRevocationList }
```

19.2.5 End-entity attribute certificate revocation list attribute type

A value of the following attribute type contains a list of revoked end-entity attribute certificates.

```
eeAttrCertificateRevocationList ATTRIBUTE ::= {
  WITH SYNTAX             CertificateList
  EQUALITY MATCHING RULE  certificateListExactMatch
  LDAP-SYNTAX             x509CertificateList.&id
  LDAP-NAME               {"EEAttrCertificateRevocationList"}
  LDAP-DESC               "X.509 EEAttr certificate revocation list"
  ID                      id-at-eeAttrCertificateRevocationList }
```

19.2.6 AA certificate revocation list attribute

The following attribute contains a list of revoked attribute certificates issued to AAs. These lists may be stored in the directory entry of the issuing authority or other directory entry (e.g., a distribution point).

```
attributeAuthorityRevocationList ATTRIBUTE ::= {
  WITH SYNTAX             CertificateList
  EQUALITY MATCHING RULE  certificateListExactMatch
  LDAP-SYNTAX             x509CertificateList.&id
  LDAP-NAME               {"AACertificateRevocationList"}
  LDAP-DESC               "X.509 AA certificate revocation list"
  ID                      id-at-attributeAuthorityRevocationList }
```

19.2.7 Delegation path attribute

The delegation path attribute contains delegation paths, each consisting of a sequence of attribute certificates.

```
delegationPath ATTRIBUTE ::= {
  WITH SYNTAX  AttCertPath
  ID           id-at-delegationPath }

AttCertPath ::= SEQUENCE OF AttributeCertificate
```

This attribute can be stored in the AA directory entry and would contain some delegation paths from that AA to other AAs. This attribute, if used, enables a more efficient retrieval of delegated attribute certificates that form frequently used delegation paths. As such, there are no specific requirements for this attribute to be used and the set of values that are stored in the attribute is unlikely to represent the complete set of delegation paths for any given AA.

19.2.8 Privilege policy attribute

The privilege policy attribute contains information about privilege policies.

```
privPolicy ATTRIBUTE ::= {
  WITH SYNTAX  PolicySyntax
  ID           id-at-privPolicy }
```

The `policyIdentifier` component includes the object identifier registered for the particular privilege policy.

If `content` is present, the complete content of the privilege policy is included.

If `pointer` is present, the `name` component references one or more locations where a copy of the privilege policy can be located. If the `hash` component is present, it contains a HASH of the content of the privilege policy that should be found at a referenced location. This hash can be used to perform an integrity check of the referenced document.

19.2.9 Protected privilege policy attribute

The protected privilege policy attribute contains privilege policies, protected within attribute certificates.

```
protPrivPolicy ATTRIBUTE ::= {
  WITH SYNTAX             AttributeCertificate
  EQUALITY MATCHING RULE  attributeCertificateExactMatch
  ID                      id-at-protPrivPolicy }
```

Note that unlike typical attribute certificates, those within the `protPrivPolicy` attribute contain privilege policies, not privileges. The issuer and holder components of these attribute certificates identify the same entity. The attribute that is included in the attribute certificate contained within the `protPrivPolicy` attribute is either the `privPolicy` attribute or the `xmlPrivPolicy` attribute.

19.2.10 XML Protected privilege policy attribute

The XML protected privilege policy attribute contains XML encoded privilege policy information.

```
xmlPrivPolicy ATTRIBUTE ::= {
  WITH SYNTAX  UTF8String -- XML-encoded privilege policy information
  ID           id-at-xmlPrivPolicy }
```

19.3 PMI general directory matching rules

This subclause defines matching rules for PMI directory attributes.

19.3.1 Attribute certificate exact match

The attribute certificate exact match rule compares for equality a presented value with an attribute value of type **`AttributeCertificate`**.

```
attributeCertificateExactMatch MATCHING-RULE ::= {
  SYNTAX  AttributeCertificateExactAssertion
  ID      id-mr-attributeCertificateExactMatch }

AttributeCertificateExactAssertion ::= SEQUENCE {
  serialNumber  CertificateSerialNumber,
  issuer        AttCertIssuer,
  ... }
```

This matching rule returns TRUE if the components in the attribute value match those in the presented value.

19.3.2 Attribute certificate match

The attribute certificate matching rule compares a presented value with an attribute value of type **`AttributeCertificate`**. This matching rule allows more complex matching than the **`certificateExactMatch`**.

```
attributeCertificateMatch MATCHING-RULE ::= {
  SYNTAX  AttributeCertificateAssertion
  ID      id-mr-attributeCertificateMatch }

AttributeCertificateAssertion ::= SEQUENCE {
  holder             [0]  CHOICE {
    baseCertificateID  [0]  IssuerSerial,
    holderName         [1]  GeneralNames,
    ...} OPTIONAL,
  issuer             [1]  GeneralNames OPTIONAL,
  attCertValidity    [2]  GeneralizedTime OPTIONAL,
  attType            [3]  SET OF AttributeType OPTIONAL,
  ... }
-- At least one component of the sequence shall be present
```

The matching rule returns TRUE if all of the components that are present in the presented value match the corresponding components of the attribute value, as follows:

– **`baseCertificateID`** matches if it is equal to the **`IssuerSerial`** component of the stored attribute value;

– **`holderName`** matches if the stored attribute value contains the name extension with the same name type as indicated in the presented value;

– **`issuer`** matches if the stored attribute value contains the name component of the same name type as indicated in the presented value;

– **`attCertValidity`** matches if it falls within the specified validity period of the stored attribute value; and

– for each **`attType`** in the presented value, there is an attribute of that type present in the **`attributes`** component of the stored value.

19.3.3 Holder issuer match

The attribute certificate holder issuer match rule compares for equality a presented value of the holder and/or issuer components of a presented value with an attribute value of type **`AttributeCertificate`**.

```
holderIssuerMatch MATCHING-RULE ::= {
  SYNTAX  HolderIssuerAssertion
  ID      id-mr-holderIssuerMatch }

HolderIssuerAssertion ::= SEQUENCE {
  holder  [0]  Holder OPTIONAL,
  issuer  [1]  AttCertIssuer OPTIONAL,
  ... }
```

This matching rule returns TRUE if all the components that are present in the presented value match the corresponding components of the attribute value.

19.3.4 Delegation path match

The **delegationPathMatch** match rule compares for equality a presented value with an attribute value of type **delegationPath**. A privilege verifier may use this matching rule to select a path beginning with a certificate issued by its SOA and ending with a certificate issued to the AA that issued the end-entity holder certificate being validated.

```
delegationPathMatch MATCHING-RULE ::= {
  SYNTAX  DelMatchSyntax
  ID      id-mr-delegationPathMatch }

DelMatchSyntax ::= SEQUENCE {
  firstIssuer  AttCertIssuer,
  lastHolder   Holder,
  ... }
```

This matching rule returns TRUE if the presented value in the **firstIssuer** component matches the corresponding elements of the **issuer** component of the first certificate in the **SEQUENCE** in the stored value and the presented value in the **lastHolder** component matches the corresponding elements of the **holder** component of the last certificate in the **SEQUENCE** in the stored value. This matching rule returns FALSE if either match fails.

19.3.5 Extension presence match

The extension presence match rule compares for equality a presented object identifier value, identifying a particular extension, with the **extensions** component of a certificate.

```
extensionPresenceMatch MATCHING-RULE ::= {
  SYNTAX  EXTENSION.&id
  ID      id-mr-extensionPresenceMatch }
```

This matching rule returns TRUE if the certificate contains the particular extension.

SECTION 4 – COMMUNICATIONS CAPABILITIES

20 Protocol support for public-key and privilege management infrastructures

The purpose of this clause to provide a specification for the transport a general wrapper protocol for the transport of PKI and/or PMI data.

20.1 General syntax

The PKI-PMI-Wrapper protocol adds authentication, integrity and optionally confidentiality to protocol data units (PDUs) used for supporting a PKI and/or a PMI. The wrapping protocol allows inclusion of any type of PDU.

A wrapped PDU is identified by an instance of the following information object class:

```
WRAPPED-PDU ::= TYPE-IDENTIFIER
```

The **WRAPPED-PDU** information object class is equivalent to the ASN.1 built information object class **TYPE-IDENTIFIER**. This information object is used to bind the type of wrapped PDU identified by an object identifier to the abstract syntax of that PDU.

The syntax of the PKI-PMI-Wrapper protocol is given by the following data type

```
PDU-wrapper ::= SIGNED{TBSPDU-wrapper}

TBSPDU-wrapper ::= SEQUENCE  {
  version                 Version DEFAULT v1,
  signatureAlgorithm      AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
  certPath            [0] IMPLICIT PkiPath,
  signedAttrs         [1] IMPLICIT SignedAttributes OPTIONAL,
  conf                    CHOICE {
    clear                   WrappedPDUInfo,
    protected               EncryptedInfo,
   ... },
  ... }

SupportedSignatureAlgorithms ALGORITHM ::= {...}

SignedAttributes ::= SET SIZE (2..MAX) OF Attribute{{SupportedSignedAttributes}}

SupportedSignedAttributes ATTRIBUTE ::= { contentType | messageDigest, ... }
```

The **version** component shall take the value **v1** or be omitted.

The **signatureAlgorithm** component shall specify the signature algorithm used for the digital signing from the set of applicable signature algorithms as defined by the **SupportedSignatureAlgorithms** information object set.

NOTE 1 – This Specification does not mandate a specific set of signature algorithms. Reference specifications or implementers' agreements may replace the dots with a set of signature algorithms to be supported for a specific environment.

NOTE 2 – By including this component, the signature algorithm is protected by the signature. In addition, the hashing by the recipient may be performed in a single pass, as the hashing algorithm is known at an early stage.

The **certPath** component shall hold the certification path necessary to verify the digital signature given in the **signature** component.

The **signedAttrs** component shall hold a list of signed attributes as defined by an instance of the **SignedAttributes** data type. Attributes of the **contentType** and **messageDigest** attribute types shall be included.

The **conf** component has two alternatives:

a) The **clear** alternative shall be taken if confidentiality (encryption) is not required and shall then include an instance of the **WrappedPDUInfo** data type (see clause 20.2).

b) The **protected** alternative shall be taken if confidentiality is required and shall then include an instance of the **EncryptedInfo** data type (see clause 20.3).

20.2 Wrapping of non-encrypted protocol data units

The syntax of the wrapped PDU information is specified using the following data type:

```
WrappedPDUInfo ::= SEQUENCE {
  pduType      WRAPPED-PDU.&id ({SupportedPduSet}),
  pduInfo      WRAPPED-PDU.&Type ({SupportedPduSet}{@pduType}),
  ... }

SupportedPduSet WRAPPED-PDU ::= {...}
```

The **pduType** component shall specify the type of PDU being wrapped.

The **pduInfo** components shall hold an instance of the wrapped PDU type.

20.3 Wrapping of encrypted protocol data unit

20.3.1 Use of the Diffie-Hellman key agreement method

To encrypt a wrapped PDU requires the establishment of shared symmetric keys. Two types of symmetric keys are defined. One type, called the content encryption key and the other type is used to encrypt (wrap) the content encryption key and is called the key-encryption key. The key agreement technique, known as the Diffie-Hellman (DH) key agreement method is used by this Specification to establish the key encryption key. This method results in a shared secret that may be used for keying material that allows generation of shared, symmetric keys. The DH method has two modes of operation relevant for this Specification. These modes are the ephemeral-static mode and the static-static mode.

The ephemeral-static mode requires that the recipient have a public-key certificate with a DH public key as certified by the issuing CA. This public-key certificate for the recipient shall be available to the sender. The sender creates a new DH key pair for each wrapped PDU it sends. In the way, the shared secret becomes different for each message.

The static-static mode requires each of the communicating entities to have a certified DH public-key certificate. In this mode, the same-shared secret number is created for each wrapped PDU. Therefore, some random user keying material has to be supplied by the sender to make different keying material for each PDU.

Both methods require both entities in a communication to have the certified end-entity public-key certificate of its partner, as communication goes in both directions.

From the shared secret, key material may be generated as specified by IETF RFC 2631. This key material is then used to create a so-called key encryption key. This key is used to encrypt a content encryption key (key wrapping). This content encryption key is generated by the sender. The key wrapping uses a different algorithm than the one used for ordinary encryption.

NOTE – The key wrapping algorithms used for AES key encryption may be found in IETF RFC 3394. These key wrapping algorithms are also listed in Annex B.

20.3.2 Encryption information syntax

The encryption information syntax is defined as:

```
EncryptedInfo ::= SEQUENCE {
  keyAgreement       KeyAgreement OPTIONAL,
  encryptedPduInfo   EncryptedPduInfo,
  ... }
```

The **keyAgreement** component, when present, shall provide the necessary information to perform the Diffie-Hellman exchange followed by the generation of the key encryption key for wrapping the content encryption key. This is detailed in clause 20.3.3. This component may be absent, if the key encryption key is not required to be renewed. Otherwise, this component shall be present.

The **encryptedPduInfo** component shall hold an instance of the **EncryptedPduInfo** data type as detailed in clause 20.3.5.

20.3.3 Key agreement specification

```
KeyAgreement ::= SEQUENCE {
  senderDhInfo          [0] SenderDhInfo,
  keyEncryptionAlgorithm SEQUENCE {
    algorithm    ALGORITHM.&id ({SupportedKeyEncryptionAlgorithm}),
    parameters   ALGORITHM.&Type({SupportedKeyEncryptionAlgorithm}{@.algorithm}),
    ... },
  ... }

SupportedKeyEncryptionAlgorithm ALGORITHM ::= {...}

SenderDhInfo ::= CHOICE {
  senderStaticInfo    [0] SenderStaticInfo,
  senderDhPublicKey   [1] SenderDhPublicKey,
  ... }

SenderStaticInfo::= SEQUENCE {
  issuer       Name,
  serialNumber CertificateSerialNumber,
  partyAinfo   UserKeyingMaterial,
  ... }

SenderDhPublicKey ::= SEQUENCE {
  algorithm   AlgorithmIdentifier {{SupportedDHPublicKeyAlgorithms}},
  publicKey   BIT STRING,
  ... }

EncryptedKey ::= OCTET STRING

SupportedDHPublicKeyAlgorithms ALGORITHM ::= {...}
```

An instance of the **KeyAgreement** data type has the following components:

The **senderDhInfo** components shall hold an instance of the **SenderDhInfo** data type, which has two alternatives:

a) The **SenderStaticInfo** alternative shall be taken if the Diffie-Hellman static-static mode is used. It shall identify the sender's DH public-key certificate and additional keying material. It shall have the following components:

– the **issuer** component shall hold the value of the **issuer** component of the sender's DH public-key certificate;

– the **serialNumber** component shall hold the value of the **serialNumber** component of the sender's DH public-key certificate; and

– the **partyAinfo** component shall hold a random octet-string of additional keying material to ensure different key encryption keys for each generation.

b) The **senderDhPublicKey** alternative shall be taken if the Diffie-Hellman ephemeral-static mode is used. It shall have the following components:

– the **algorithm** component shall identify the DH agreement algorithm;

_ the **publicKey** components shall hold the generated DH public-key of the sender.

20.3.4 Generation of keying material

IETF RFC 2631 specifies how keying material is generated from the developed shared secret. It is repeated here for easy reference.

The keying material is produced by providing a hash using the SHA-1 algorithms of the shared secret concatenated with the DER encoding of an instance of the **OtherInfo** data type below. The SHA-1 produces 160 bits of keying material. If that is insufficient, the hash operation has to be repeated until sufficient keying material has been produced. The leftmost part (most significant bits) of the keying material is then used as the key-encryption key.

```
OtherInfo ::= SEQUENCE {
  keyInfo            KeySpecificInfo,
  partyAInfo    [0] EXPLICIT OCTET STRING OPTIONAL,
  suppPubInfo   [2] EXPLICIT OCTET STRING }

KeySpecificInfo ::= SEQUENCE {
  algorithm ALGORITHM.&id ({SupportedKeyEncryptionAlgorithm}),
  counter   OCTET STRING SIZE (4) }
```

The **OtherInfo** data type has the following components:

a) The **keyInfo** component shall hold an instance of the **KeySpecificInfo** data type with the following the components:

– The **algorithm** component shall specify the object identifier for the key encryption algorithm, i.e., it shall have the same value as specified for the **KeyAgreement** data type above.

– The **counter** component shall have the hex value 00 00 00 01 for the first pass for generating keying material for a specific key and incremented by one for each additional path.

b) The **partyAInfo** component shall be present if the **senderStaticInfo** alternative was taken for the **SenderDhInfo** data type above. Otherwise, this component shall be absent. If present, it shall have the value supplied in the instance of the **SenderStaticInfo** data type.

c) The **suppPubInfo** shall the length of the generated key encryption key expressed as a 4 octet hex number.

NOTE – If an AES-256 key is generated, the value is then 00 01 00'H.

20.3.5 Encryption encoding

The encryption information syntax is defined as:

```
EncryptedPduInfo ::= SEQUENCE {
  pduType                  WRAPPED-PDU.&id ({SupportedPduSet}),
  encryptedKey             EncryptedKey OPTIONAL,
  pduEncryptionAlgorithm   SEQUENCE {
    algorithm                ALGORITHM.&id ({SymmetricEncryptionAlgorithms}),
    parameter                ALGORITHM.&Type
                 ({SymmetricEncryptionAlgorithms}{@.algorithm})} OPTIONAL,
  encryptedPdu           [0] EncryptedPdu,
  ... }

EncryptedKey ::= OCTET STRING

SymmetricEncryptionAlgorithms ALGORITHM ::= {...}

EncryptedPdu ::= OCTET STRING
```

The **EncryptedPduInfo** data type has the following components:

a) The **encryptedKey** component shall hold the encrypted content encryption key.

b) The **pduType** component shall specify the type of PDU to be encrypted and transmitted.

c) The **pduEncryptionAlgorithm** component has the following components:

– the **algorithm** component shall specify the identity of the symmetric encryption algorithm; and

– the **parameter** component shall include the parameters associated with the symmetric encryption algorithm, if any.

d) The **EncryptedPdu** component shall hold the encrypted PDU.

20.4 Check of PKI-PMI-Wrapper protocol elements

Several checks are independent of the wrapped PDUs. Error codes are defined for different exception conditions for the PKI-PMI-Wrapper protocol elements. Such error information shall be returned by the wrapped protocol as part of its exception handling.

20.4.1 General checking

It should be checked:

a) whether the **version** component is specifying an unsupported version and if so, return a **unsupportedWrapperVersion** error code;

b) whether the signature algorithm is supported and if not, return an **unsupportedSignatureAlgorithm** error code;

c) whether the certificates component contains public-key certificates sufficient to create a certification path and if not return an **incompleteCertPath** error code;

d) whether the certification path validates and if not, return a **certificationPathFailure** error code;

e) whether the digital signature is valid and if not, return an **invalidSignature** error code;

f) whether mandatory attributes for the wrapped PDU are included in the **signedAttrs** component and if not, return a **missingMandatoryAttributes** error code;

g) whether unwanted attributes for the wrapped PDU are included in the **signedAttrs** component and if so, return a **unwantedAttribute** error code;

20.4.2 Specific checking when not encrypting the wrapped PDU

If the **clear** alternative of the **TBSPDU-wrapper** data type is taken for the **conf** component, then for the **WrappedPDUInfo** data type check:

a) whether the **`pduType`** component specifies a supported type and if not, return an **`unsupportedPduType`** error code;

b) whether the **`pduType`** component specifies a PDU out of sequence and if so, return an **`unexpectedPduType`** error code.

20.4.3 Specific checking when encrypting the wrapped PDU

If the **`protected`** alternative of the **`TBSPDU-wrapper`** data type is taken for the **`conf`** component, then the actions specified in 20.4.3.1 shall be taken and if encryption is selected, then the actions specified in 20.4.3.2 shall also be taken.

20.4.3.1 Checking of the key agreement specification

The **`senderDhInfo`** component of the value of the **`KeyAgreement`** data type shall be checked as follows:

a) if the **`senderStaticInfo`** component is taken, then:

– check whether the **`issuer`** and **`serialNumber`** components together identify an available DH public-key certificate for the sender and if not, return an **`unknownDHpkCetificate`** error code,

– check whether the **`partyAinfo`** component holds 64 octets of random keying material and if not, return an **`invalidKeyingMaterial`** error code;

b) if the **`senderDhPublicKey`** alternative is taken, then:

– check whether the **`algorithm`** component identifies a DH public-key algorithm that is equal to the one specified by the DH public-key certificate held by the recipient and if not, return **`dhAlgorithmMismatch`** error code,

– check whether the **`publicKey`** component holds a valid DH public-key and if not, return an **`invalideDhPublickey`** error code;

The **`keyEncryptionAlgorithm`** component of the value of the **`KeyAgreement`** data type shall be checked as follows:

a) check whether the **`algorithm`** component identifies a supported key wrapping algorithm and if not, return an **`unsupportedKeyWrappingAlgorithm`** error code;

b) check whether the presence or absent of the **`parameters`** component is as according to the type of algorithm and if not, return a **`keyEncAlgorithmParametersMissing`** error code or **`keyEncAlgorithmParametersNotAllowed`** error code, as appropriate.

20.4.3.2 Checking of the encrypted PDU information

The value of the **`EncryptedPduInfo`** data type shall be checked

a) whether the **`pduType`** component specifies a supported PDU type and if not, returns an **`unsupportedPduType`** error code;

b) whether the **`pduType`** component specifies a PDU out of sequence and if so, returns an **`unexpectedPduType`** error code;

c) whether the **`pduEncryptionAlgorithm`** components identify a supported symmetric encryption algorithm and if not, return an **`unsupportedSymmetricKeyAlgorithm`** error code;

d) whether the presence or absence of the **`parameter`** component of the **`pduEncryptionAlgorithm`** component is as according to the type of algorithm and if not, returns a **`keySymAlgorithmParametersMissing`** error code or **`keyEncAlgorithmParametersNotAllowed`** error code, as appropriate.

e) whether the parameter component of the **`pduEncryptionAlgorithm`** component, if present, has the right content and if not, returns the **`invalidParmsForSymEncryptAlgorithms`** error code;

f) whether the **`encryptedPdu`** component holds a valid PDU after decryption by the decrypted content encryption key and if not, returns **`decryptionFailed`** error code;

g) whether the decrypted PDU is of a type corresponding the type specification in the **`pduType`** component and if not, returns an **`invalidPduSyntax`** error code.

20.5 PKI-PMI-Wrapper error codes

The following error codes are defined the PKI-PMI-Wrapper:

```
PkiWaError ::= ENUMERATED {
  unsupportedWrapperVersion           (0),
  unsupportedSignatureAlgorithm       (1),
  incompleteCertPath                  (2),
  certificationPathFailure            (3),
  invalidSignature                    (4),
  missingMandatoryAttributes          (5),
  unwantedAttribute                   (6),
  unsupportedPduType                  (7),
  unexpectedPduType                   (8),
  invalidPduSyntax                    (9),
  unknownDHpkCetificate               (10),
  invalidKeyingMaterial               (11),
  dhAlgorithmMismatch                 (12),
  invalideDhPublickey                 (13),
  unsupportedKeyWrappingAlgorithm     (14),
  keyEncAlgorithmParametersMissing    (15),
  keyEncAlgorithmParametersNotAllowed (16),
  invalidParmsForSymEncryptAlgorithms (17),
  decryptionFailed                    (18),
  ... }
```

21 Authorization and validation list management

21.1 General

To support the creation and maintenance of authorization and validation lists (AVLs), communications between involved entities is required.

AVL management may involve entities constrained with respect to processing power, storage, bandwidth, etc. In this environment, protocol overhead should be kept to a minimum.

This specification provides specification for two protocols for AVL management. The authorization and validation management protocol (AVMP) is used between an authorizer and an AVL entity for AVL management. The CA subscription protocol (CASP) is used between an authorizer and a CA at which the authorizer subscribes to public-key certificate status. The CASP protocol may also be used by a relying party that requires revocation information through a subscription service with appropriate CAs. Each of these protocols comprises a number of PDU types.

21.2 Defined protocol data unit (PDU) types

The following PDU types are defined for the AVMP and the CASP:

```
AvlPduSet WRAPPED-PDU ::= {
  certReq |
  certRsp |
  addAvlReq |
  addAvlRsp |
  replaceAvlReq |
  replaceAvlRsp |
  deleteAvlReq |
  deleteAvlRsp |
  rejectAVL |
  certSubscribeReq |
  certSubscribeRsp |
  certUnsubscribeReq |
  certUnsubscribeRsp |
  certReplaceReq |
  certReplaceRsp |
  rejectCAsubscribe,
  ... }
```

21.3 Checking of received PDU

When a message is received, the wrapping part of the message shall be checked as specified in clause 20.4. If an exception condition is encountered, the resulting error code shall be returned to the sender. If the checks specified in clause 20.4 have been completed without detecting an exception condition, the recipient shall perform a number of validation steps on the wrapped PDU. If this e validation fails at any step, no further validation is necessary and the recipient shall return an appropriate error code. Error codes for the WLMP and CASP are specified in clauses 21.4.7 and 21.5.8.

A number of checks are common across different PDU types. It shall be checked:

a) whether the **version** component of the PDU is supported and if not, returns an **unsupportedAVMPversion** or **unsupportedCASPversion** error code, as appropriate;

b) whether the **sequence** component holds a valid sequence number and if not, returns a **sequenceError** error code as specified in clause 21.4.7 for the WLMP and in clause 21.5.8 for CASP.

21.4 Authorization and validation management protocol

21.4.1 Authorization validation management introduction

The authorization and validation management is concerned with how the authorizer maintains AVL information within the AVL entities it supports. It comprises a set of PDU exchanges as detailed in the following.

It shall be the same trusted CA that has certified the end-entity public-key certificates for the authorizer and the AVL entities it serves. Each AVL entity shall be supplied with:

– its own end-entity public-key certificate and corresponding private key to be used for digital signature generation;

– if encryption is required for some types of communications with that AVL entity, its DH end-entity public-key certificate and corresponding private key used for key agreement according to the Diffie-Hellman method; and

– the CA certificate of the issuing CA for above two end-entity public-key certificates.

21.4.2 Authorization and validation management protocol common components

Some components are common across different content types. The **WLMPcommonComponents** data type comprises these components.

```
AVMPcommonComponents ::= SEQUENCE {
  version    AVMPversion DEFAULT v1,
  timeStamp  GeneralizedTime,
  sequence   AVMPsequence,
  ... }

AVMPversion ::= ENUMERATED { v1(1), v2(2), v3(3), ... }

AVMPsequence ::= INTEGER (1..MAX)
```

The **AVMPcommonComponents** data type has the following components.

a) The **version** component shall hold the version of the AVMP. The current version is version **v1**.

b) The **timeStamp** component shall be the GMT (UTC) generalized time at which the content was created.

c) The **sequence** component shall take the value 0 for the first message sent for a given value of the **timeStamp** component and be incremented by one for each message sent for the same value of the **time** component. When the time component change from sending one message to sending the next, the sequence number shall restart at the value 0. The purpose is:

   – to allow detecting of replay of messages caused by an error or by an adversary;

   – to pair requests and responses; and

   – to detect missing messages.

21.4.3 Public-key certificate management

```
certReq WRAPPED-PDU ::= {
               CertReq
  IDENTIFIED BY id-certReq }

CertReq ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  ... }
```

The **certReq** PDU type is used by an AVL entity to initialization information allowing that AVL entity to participate in communication with the authorizer.

Instances of this PDU type shall be sent without encryption. The signature shall be generated using the private key of the AVL entity.

```
CertReq ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  ... }
```

The **CertReq** data type specifies the syntax of the wrapped PDU content and has the following components:

a) The components of the **AVMPcommonComponents** data type is specified in clause 21.5.2. The **sequence** component shall take the value 0.

The authorizer shall check the validity of the request:

a) by checking as specified in clause 21.4;

b) by checking whether the sender is a recognized AVL entity and if not return an **unknownAVLentity** error code.

```
certRsp WRAPPED-PDU ::= {
              CertRsp
  IDENTIFIED BY id-certRsp }
```

The authorizer shall use the **certRsp** PDU type to respond to the certificate request.

```
CertRsp ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  result        CHOICE {
    success       [0]  CertOK,
    failure       [1]  CertErr,
    ... },
  ... }

CertOK ::= SEQUENCE {
  dhCert  Certificate,
  ... }

CertErr ::= SEQUENCE {
  notOK  CHOICE {
    wrErr   PkiWrErr,
    avmpErr AVMP-error,
    ... },
  note   Notification OPTIONAL,
  ... }
```

The **CertRsp** data type specifies the syntax of the wrapped PDU and has the following components:

a) The components of the **AVMPcommonComponents** data type is specified in clause 21.5.2. The **sequence** component shall take the value 0.

b) The **result** component has the following two alternatives:

– The **success** alternative shall be taken if the request was accepted. The **dhCert** component shall hold the DH end-entity public-key certificate owned by the authorizer.

– The **failure** alternative shall be taken if the request fails. It shall have the following component:

The **AVMP-error** data type as specified in clause 21.4.7.

21.4.4 Add authorization and validation list

```
addAvlReq CONTENT-TYPE ::= {
              AddAvlReq
  IDENTIFIED BY id-addAvlReq }
```

The authorizer uses the **addAvlReq** content type to initiate the addition of an AVL to an AVL entity.

```
AddAvlReq ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  certlist      CertAVL,
  ... }
```

The **AddAvlReq** data type specifies the syntax of the actual content and has the following components:

a) The components of the **AVMPcommonComponents** data type are specified in clause 21.5.2. The **sequence** component shall take the value 1.

b) The **certList** component shall hold the AVL to be added to the AVL entity.

The recipient AVL entity shall check the validity of the request:

a) by checking as specified in clause 21.4;

b) by checking the validity of the signature on the received AVL and if invalid, return an **invalidAvlSignature** error code;

c) by checking whether all the AVL mandatory components are present and if not, return a **missingAvlComponent** error code;

d) by checking whether the **version** component on the received AVL validation list specifies a supported version and if not, return an **invalidAvlVersion** error code;

e) if the **serialNumber** component is present in the received AVL, then check whether an AVL with the same value already exists and if so, return a **duplicateAVL** error code;

f) if the **serialNumber** component is absent in the received AVL, then check whether an AVL with absent **serialNumber** component already exists and if so, return a **duplicateAVL** error code;

g) by checking whether the **constrained** component of the received AVL corresponds to the capabilities of the AVL entity and if not, return a **constrainedRequired** error code or a **nonConstrainedRequired** error code, as appropriate;

h) for each element of the **entries** component of the received AVL:

– by checking whether the **idType** component contains a valid alternative and if not, return a **protocolError** error code;

– if the **certIdentifier** alternative is chosen, then:

i) by checking whether the **certIdentifier** component contains a valid alternative and if not , return a **protocolError**,

ii) if the **entityGroup** alternative is taken and the **constrained** component has the value **TRUE**, return a **notAllowedForConstrainedAVLEntity** error code;

– by checking whether the **scope** component holds some unsupported scope restrictions, an if so, return an **unsupportedScopeRestriction** error code;

– by checking whether the **entryExtensions** component contains an unsupported critical extension and if so, return an **unsupportedCriticalEntryExtension** error code;

j) by checking whether the AVL contains an unsupported critical extension and if so, return an **unsupportedCriticalExtension** error code;

k) by checking whether the maximum number of AVLs has been exceeded by the new AVL and if so, return a **maxAVLsExceeded** error code.

NOTE – Maximum limit might be just a single AVL.

```
addAvlRsp CONTENT-TYPE ::= {
                AddAvlRsp
  IDENTIFIED BY  id-addAvlRsp }
```

The recipient AVL entity shall use the **addAvlRsp** content type to report the outcome of an add AVL

```
AddAvlRsp ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  result        CHOICE {
    success        [0]  AddAvlOK,
    failure        [1]  AddAvlErr,
    ... },
  ... }

AddAvlOK ::= SEQUENCE {
  ok      NULL,
  ... }

AddAvlErr ::= SEQUENCE {
  notOK  AVMP-error,
  ... }
```

The **AddAvlRsp** data type specifies the actual content and has the following components:

a) The components of the **AVMPcommonComponents** data type is specified in clause 21.5.2.

b) The **result** component has the following two alternatives:

– The **success** alternative shall be taken if the addition of an AVL was performed successfully;

– The **failure** alternative shall be taken if the addition of an AVL failed. The **AVMP-error** data type is specified in clause 21.5.7.

21.4.4 Replace authorization and validation list

```
replaceAvlReq CONTENT-TYPE ::= {
                 ReplaceAvlReq
  IDENTIFIED BY  id-replaceAvlReq }
```

The authorizer uses the **replaceAvlReq** content type to initiate the replacement of an AVL at an AVL entity. It shall be used when changes to the AVL have occurred.

```
ReplaceAvlReq ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  old           AvlSerialNumber OPTIONAL,
  new           CertAVL,
  ... }
```

The **ReplaceAvlReq** data type specifies the actual content and has the following components:

a) the components of **AVMPcommonComponents** data type as specified in clause 21.5.2;

b) the **old** component, when present, shall hold the serial number of the old AVL and it shall be absent if the authorizer expects that an AVL with no sequence number exits; and

c) the **new** component shall hold the replacement AVL.

The AVL entity shall verify the validity of the request by checking:

a) as specified in 21.5.3 items a) to j);

b) if the **old** component was present in the request, then check whether the **AvlSerialNumber** value specified in that component matches the **AvlSerialNumber** of a local authorization validation list and if not, return an **unknownAvl** error code;

c) if the **old** component was absent in the request, then check whether there locally is just a single AVL and that AVL is without the **serialNumber** component and if not, return an **unknownAvl** error code.

```
replaceAvlRsp CONTENT-TYPE ::= {
                 ReplaceAvlRsp
  IDENTIFIED BY  id-replaceAvlRsp }
```

The AVL entity shall use the **replaceAvlRsp** content type to report the outcome of an AVL replace request.

```
ReplaceAvlRsp ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  result        CHOICE {
    success       [0]  RepAvlOK,
    failure       [1]  RepAvlErr,
    ... },
  ... }

RepAvlOK ::= SEQUENCE {
  ok       NULL,
  ... }

RepAvlErr ::= SEQUENCE {
  notOK    AVMP-error,
  ... }
```

The **ReplaceAvlRsp** data type specifies the syntax of the actual content and has the following components:

– The components of the **AVMPcommonComponents** data type as specified in clause 21.5.2.

– The **result** component has the following two alternatives:

  a) The **success** alternative shall be taken if the replacement of an AVL was performed successfully,

  b) The **failure** alternative shall be taken if the replacement of an AVL failed. The **AVMP-error** data type is specified in clause 21.5.7.

21.4.5 Delete authorization and validation list

```
deleteAvlReq CONTENT-TYPE ::= {
                 DeleteAvlReq
  IDENTIFIED BY  id-deleteAvlReq }
```

The authorizer uses the **deleteAvlReq** content type to initiate deletion of an AVL at an AVL entity.

```
DeleteAvlReq ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  avl-Id        AVLSerialNumber OPTIONAL,
  ... }
```

The **DeleteAvlReq** data type specifies the syntax of the actual content and has the following components:

a) The components of **AVMPcommonComponents** data type as specified in clause 21.5.2.

b) The **avl-Id** component, when present, shall identify the AVL to be deleted.

The recipient AVL entity shall verify the validity of the request by checking

a) as specified in clause 21.4;

b) if the **avl-id** component was present in the request, then check whether the **AlvSerialNumber** value specified in that component matches the **AvlSerialNumber** of a local AVL and if not, return an **unknownAVL** error code;

c) if the **avl-id** component was absent in the request, then check whether there locally is just a single AVL and that AVL is without the **serialNumber** component and if not, return an **unknownAVL** error code.

```
deleteAvlRsp CONTENT-TYPE ::= {
                 DeleteAvlRsp
  IDENTIFIED BY  id-deleteAvlRsp }
```

The recipient AVL entity shall use the **deleteAvlRsp** content type to report the outcome of a delete AVL request.

```
DeleteAvlRsp ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  result        CHOICE {
    success       [0]  DelAvlOK,
    failure       [1]  DelAvlErr,
    ... },
  ... }

DelAvlOK ::= SEQUENCE {
  ok      NULL,
  ... }

DelAvlErr ::= SEQUENCE {
  notOK  AVMP-error,
  ... }
```

The **DeleteAvlRsp** data type specifies the syntax of the actual content. It has the following components:

The components of the **AVMPcommonComponents** data type as specified in clause 21.5.2.

The **result** component has the following two alternatives:

a) The **success** alternative shall be taken if the deletion of an AVL was performed successfully.

b) The **failure** alternative shall be taken if the deletion of an AVL failed. The **AVMP-error** data type is specified in clause 21.5.7.

21.4.6 Authorization and validation list reject

```
rejectAVL  CONTENT-TYPE ::= {
                 RejectAVL
  IDENTIFIED BY  id-rejectAVL }
```

The **rejectAVL** content type is used by the authorizer to report problems with a response from the AVL entity.

```
RejectAVL ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  reason        AVMP-error,
  ... }
```

The **RejectAVL** data type specifies the syntax of the actual content and has the following components:

a) The **sequence** component of the **AVMPcommonComponents** data type shall take the same value as in the response on which it is reporting.

b) The **AVMP-error** as specified in clause 21.5.7.

The authorizer shall verify the validity of a received response by checking

a) as specified in clause 21.4.

21.4.7 Authorization and validation list error codes

A value of the **AVMP-error** data type is used by an AVL entity to report an error when processing a request from the authorizer. It is also used by an authorizer to reject a faulty response from an AVL entity.

```
AVMP-error ::= ENUMERATED {
  noReason                          (0),
  unknownAvlEntity                  (1),
  unknownContentType                (2),
  unsupportedAVMPversion            (3),
  missingContent                    (4),
  missingContentComponent           (5),
  invalidContentComponent           (6),
  sequenceError                     (7),
  protocolError                     (8),
  invalidAvlSignature               (9),
  duplicateAVL                      (10),
  missingAvlComponent               (11),
  invalidAvlVersion                 (12),
  notAllowedForConstrainedAVLEntity (13),
  constrainedRequired               (14),
  nonConstrainedRequired            (15),
  unsupportedCriticalEntryExtension (16),
  unsupportedCriticalExtension      (17),
  maxAVLsExceeded                   (18),
  unknownCert                       (19),
  unknownAVL                        (20),
  unsupportedScopeRestriction       (21),
  ... }
```

The following authorization and validation list error codes are defined:

a) the **noReason** value shall be selected when no other error code is applicable;

b) the **unknownContentType** value shall be selected if the content type is not known by the receiver;

c) the **unsupportedAVMPversion** value shall be selected if a request or response content specified an AVMP version not supported;

d) the **missingContent** value shall be selected when the request or response did not include a content;

e) the **missingContentComponent** value shall be selected when a request or response did not include a mandatory component;

f) the **invalidContentComponent** value shall be selected when an unexpected component was included in a request or response;

g) the **sequenceError** value shall be selected when:

- an end entity receives a request content of the **addAvlReq** content type that did not have the **sequence** component set to 1;
- an end entity receives a request content not of the **addAvlReq** content type that did not have the **sequence** component set to one more than for the previous request; or
- an authorizer receives a response content with a **sequence** component value different from the one in the corresponding request content;

h) the **protocolError** value shall be selected when a protocol error is encountered;

i) the **invalidSignature** value shall be selected when the signature on an AVL is invalid;

j) the **duplicateAVL** value shall be selected when the authorizer attempts to add an already existing AVL to an end entity;

k) the **missingAvlComponent** value shall be selected when a received AVL is missing a mandatory component;

l) the **invalidAvlVersion** value shall be selected when an unsupported authorization validation list version is received;

m) the **notAllowedForConstrainedAVLEntity** shall be selected if a component is included that is not allowed by a constrained AVL entity;

n) the **constraintRequired** value shall be selected when the end entity requires an AVL with the **constraint** component set to **FALSE**;

o) the **nonConstraintRequired** value shall be selected when the end entity requires an AVL with the **constraint** component set to **TRUE**;

p) the **unsupportedCriticalEntryExtension** value shall be selected when a received AVL contains an unsupported critical entry extension;

q) the **unsupportedCriticalExtension** value shall be selected when a received AVL contains an unsupported critical extension;

r) the **maxAVLsExceeded** value shall be selected when the addition of an AVL would bring the number of AVLs beyond a locally determined value;

s) the **unknownCert** value shall be selected when an unknown public-key certificate was referenced in an update request;

t) the **unknownAVL** value shall be selected when an end entity receives a content including a value of the **AvlSerialNumber** data type that did not match any local AVL;

u) the **unsupportedScopeRestriction** value shall be selected when an unsupported scope restriction was included in an AVL.

21.5 Certification authority subscription protocol

21.5.1 Certification authority subscription introduction

The CA subscription is concerned with how the authorizer maintains AVL status information by subscribing to the necessary information from relevant CAs. It is only relevant for authorizer supporting authorization validation lists for constraint end entities.

Before subscribing to maintenance information, the authorizer needs to know the exact certification configuration for the end entities it supports. The following information is necessary to establish:

a) The end-entity public-key certificates for the AVL entities for which AVL support is to be provided.

b) For each AVL entity from a), the end-entity public-key certificates for the AVL entities to which communications are possible.

c) The CA-certificates and trust anchor information necessary to establish any necessary certification path.

This Specification does not gives details on how an authorizer obtains this information. It could be by local configuration or by abstract of a centralized database.

The CASP comprises a set of CMS exchange types as detailed in the following.

21.5.2 Certification authority subscription common components

Some components are common across different content types. The **CASPcommonComponents** data type comprises these components.

```
CASPcommonComponents ::= SEQUENCE {
  version     CASPversion DEFAULT v1,
  sequence    CASPsequence,
  ... }

CASPversion ::= ENUMERATED { v1(1), v2(2), v3(3), ... }

CASPsequence ::= INTEGER (1..MAX)
```

The **CASPcommonComponents** data type has the following components.

The **version** component shall hold the version of the CASP. The current version is version **v1**.

The **sequence** component shall hold a sequence number of a message being sent. The sequence number is used:

a) to allow detection replay of messages caused by an error or by an adversary;

b) to pair requests and responses; and

c) to detect missing messages.

21.5.3 Public-key certificate subscription

```
certSubscribeReq CONTENT-TYPE ::= {
                 CertSubscribeReq
  IDENTIFIED BY  id-certSubscribeReq }
```

The authorizer uses the **certSubscribeReq** content type to request a specific CA to supply status information about public-key certificates issued by this CA relevant for the AVLs supported by the authorizer.

```
CertSubscribeReq ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  certs    SEQUENCE OF SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  ... }
```

The **CertSubscribeReq** data type specifies the syntax of the actual content and has the following components:

a) The components of the **CASPcommonComponents** data type as specified in clause 22.6.2.

b) The **certs** component shall identify a list of end-entity public-key certificates, for which the authorizer requests information about status changes. It has the following subcomponents for each element:

– the **subject** subcomponent shall be the name of the entity for which the end-entity public-key certificate has been issued;

– the **serialNumber** subcomponent shall be the serial number for the end-entity public-key certificate in question.

The CA shall verify the validity of the request by checking:

a) as specified in clause 21.4;

b) each element of the **certs** component for validity, i.e., whether it identifies an end-entity public-key certificate issued by the CA. If not, an **unknownCert** status code shall be returned in the corresponding element of the response.

```
certSubscribeRsp CONTENT-TYPE ::= {
                 CertSubscribeRsp
  IDENTIFIED BY  id-certSubscribeRsp }
```

The CA shall use the **certSubscribeRsp** content type to report the outcome of the subscription request.

```
CertSubscribeRsp ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  result       CHOICE {
    success      [0]  CertSubscribeOK,
    failure      [1]  CertSubscribeErr,
    ... },
  ... }

CertSubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok        [0] SEQUENCE {
    cert         Certificate,
    status       CertStatus,
    revokeReason CRLReason OPTIONAL,
    ... },
  not-ok    [1] SEQUENCE {
    status       CASP-CertStatusCode,
    ... },
  ... }

CASP-CertStatusCode ::= ENUMERATED {
  noReason                  (1),
  unknownCert               (2),
  ... }
```

```
CertStatus ::= ENUMERATED {
  good    (0),
  revoked (1),
  on-hold (2),
  expired (3),
  ... }

CertSubscribeErr ::= SEQUENCE {
  code        CASP-error,
  ... }
```

The **`CertSubscribeRsp`** data type specifies the actual content and has the following components:

The components of the **`CASPcommonComponents`** data type as specified in clause 21.6.2.

The **`result`** component has the following two alternatives:

a) The **`success`** alternative shall be taken if the subscription was accepted for at least one public-key certificate. It shall then hold a value of the **`CertSubscribeOK`** data type.

b) The **`failure`** alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned. It shall then hold a value of the **`CertSubscribeErr`** data type. The CASP-error data type is specified in clause 21.6.8.

The **`CertSubScribeOK`** shall include an element for each public-key certificate specified in the request in the same order. Each element has two alternatives:

a) The **`ok`** alternative shall be taken when public-key certificate information was successfully retrieved. It has the following components:

   – the **`cert`** component shall hold the public-key certificate for the requested subject;

   – the **`status`** component shall hold the status of the public-key certificate and shall take one the following values:

      i) the **`good`** value signals that the represented public-key certificate can be trusted,

      ii) the **`revoked`** value signals that the represented public-key certificate has been revoked and can no longer be trusted,

      iii) the **`on-hold`** value signals that the represented public-key certificate has been put on hold status and should not be trusted for the time being,

      iv) the **`expired`** value signals that the represented public-key certificate has expired and cannot longer be trusted;

   – the **`revokeReason`** component may be present when the **`status`** component is set to **`revoked`**, and shall otherwise be absent. When present, it shall indicate the reason for the revocation as defined in clause 9.5.3.1.

b) The **`not-ok`** alternative shall be taken when a corresponding public-key certificate was not identified:

   – the **`no-reason`** status code shall be returned when no other code is applicable;

   – the **`unknownCert`** status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

21.5.4 Public-key certificate un-subscription

```
certUnsubscribeReq CONTENT-TYPE ::= {
                 CertUnsubscribeReq
  IDENTIFIED BY  {id-cmsct 10} }
```

The authorizer uses the **`certUnsubscribeReq`** content type to request a specific CA to stop supplying status information about public-key certificates issued by that CA.

```
CertUnsubscribeReq ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  certs  SEQUENCE OF SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  ... }
```

The **`CertUnsubscribeReq`** data type specifies the actual content and has the following components:

a) The components of **CASPcommonComponents** data type as specified in clause 22.6.2.

b) The **certs** component shall identify a list of public-key certificates for which, the authorizer requests stop for information about status changes. It has the following subcomponents for each public-key certificate:

   – The **subject** subcomponent shall be the name of the entity to which the public-key certificate has been issued.

   – The **serialNumber** subcomponent shall be the serial number for the public-key certificate in question.

The CA shall verify the validity of the request by checking:

a) as specified in clause 21.4;

b) each element of the certs component for validity, i.e., whether it identifies a public-key certificate issued by the CA. If not, an **unknownCert** status code shall be returned in the corresponding element of the response.

```
certUnsubscribeRsp CONTENT-TYPE ::= {
                  CertUnsubscribeRsp
  IDENTIFIED BY  id-certUnsubscribeReq } }
```

The CA shall use the **certUnsubscribeRsp** content type to report the outcome of the un-subscription request.

```
certUnsubscribeRsp CONTENT-TYPE ::= {
                  CertUnsubscribeRsp
  IDENTIFIED BY  id-certUnsubscribeRsp }

CertUnsubscribeRsp ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  result       CHOICE {
    success        [0]  CertUnsubscribeOK,
    failure        [1]  CertUnsubscribeErr,
    ... },
  ... }

CertUnsubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok        [0] SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  not-ok    [1] SEQUENCE {
    status       CASP-CertStatusCode,
    ... },
  ... }

CertUnsubscribeErr ::= SEQUENCE {
  code         CASP-error,
  ... }
```

The **CertSubscribeRsp** data type specifies the actual content and has the following components:

The components of the **CASPcommonComponents** data type as specified in clause 21.6.2.

The **result** component has the following two alternatives:

a) The **success** alternative shall be taken if the subscription was accepted for at least one public-key certificate. It shall then hold a value of the **CertUnsubscribeOK** data type.

b) The **failure** alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned. It shall then hold a value of the **CertUnsubscribeErr** data type. The **CASP-error** data type is specified in clause 21.6.8.

The **CertUnsubScribeOK** includes an element for each public-key certificate specified in the request in the same order. Each element has two alternatives:

a) The **ok** alternative shall be taken when public-key certificate information was successfully retrieved. It has the following components:

   – the **subject** component shall hold the name of the subject to which the public-key certificate had been issued;

   – the **serialNumber** component shall hold the serial number for the public-key certificate.

b) The **not-ok** alternative shall be taken when a corresponding public-key certificate was not identified.

   – the **no-reason** status code shall be returned when no other status code is applicable;

– the **unknownCert** status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

The **error** alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned. The CASP-error data type is specified in clause 21.6.8.

21.5.5 Public-key certificate replacements

```
certReplaceReq CONTENT-TYPE ::= {
                 CertReplaceReq
  IDENTIFIED BY  id-certReplaceReq }
```

The CA shall use the **certReplacementReq** content type to submit replaced end-entity public-key certificates to the authorizer.

```
CertReplaceReq ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  certs   SEQUENCE OF SEQUENCE {
    old     CertificateSerialNumber,
    new     Certificate,
    ... },
  ... }
```

The **CertReplacementReq** data type specifies the actual content and has the following components:

a) The components of the **CASPcommonComponents** data type as specified in clause 21.6.2.

b) The **certs** component shall identify a list of public-key certificate replacements. It has the following subcomponents for each public-key certificate:

– The **old** subcomponent shall hold the identification of the public-key certificate to be replaced.

– The **new** subcomponent shall hold the replacement public-key certificate.

The authorizer shall verify the validity of the request by checking:

a) as specified in clause 21.4;

b) each element of the **certs** component for validity:

– whether the **old** subcomponent identifies a public-key certificate at the authorizer and if not, an **unknownCert** status code shall be returned in the corresponding element of the response;

b) each element of the **certs** component for validity, i.e., whether it identifies a public-key certificate issued by the CA. If not, an **unknownCert** status code shall be returned in the corresponding element of the response.

```
certReplaceRsp CONTENT-TYPE ::= {
                 CertReplaceRsp
  IDENTIFIED BY  id-certReplaceRsp }
```

The authorizer shall use the **certReplacementRsp** content type to report the outcome of the subscription request.

```
CertReplaceRsp ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  result        CHOICE {
    success       [0]  CertReplaceOK,
    failure       [1]  CertReplaceErr,
    ... },
  ... }

CertReplaceOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok         [0] SEQUENCE {
    issuer         Name,
    serialNumber   CertificateSerialNumber,
    ... },
  not-ok     [1] SEQUENCE {
    status         CASP-CertStatusCode,
    ... },
  ... }

CertReplaceErr ::= SEQUENCE {
  code          CASP-error,
  ... }
```

The **CertReplaceRsp** data type specifies the actual content and has the following components:

The components of the **CASPcommonComponents** data type as specified in clause 21.6.2.

The **result** component has the following two alternatives:

a) The **success** alternative shall be taken if the subscription was accepted for at least one public-key certificate. It shall then hold a value of the **CertReplaceOK** data type.

b) The **failure** alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned. It shall then hold a value of the **CertReplaceErr** data type. The CASP-error data type is specified in clause 21.6.8.

The **CertReplace** data type includes an element for each public-key certificate specified in the request in the same order. Each element has two alternatives:

a) The **ok** alternative shall be taken when public-key certificate information was successfully retrieved. It has the following components:
   - the **subject** component shall hold the name of the subject to which the public-key certificate had been issued;
   - the **serialNumber** component shall hold the serial number for the public-key certificate.

b) The **not-ok** alternative shall be taken when a corresponding public-key certificate was not identified.
   - the **no-reason** status code shall be returned when no code is applicable;
   - the **unknownCert** status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

21.5.6 End-entity public-key certificate updates

```
certUpdateReq CONTENT-TYPE ::= {
                 CertUpdateReq
  IDENTIFIED BY  id-certUpdateReq }
```

The CA shall use the **certUpdateReq** content type to submit to the authorizer updated status information on public-key certificates.

```
CertUpdateReq ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  certs   SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    subject       Name,
    serialNumber  CertificateSerialNumber,
    certStatus    CertStatus,
    ... },
  ... }
```

The **CertUpdateReq** data type specifies the actual content and has the following components:

a) The components of the **CASPcommonComponents** data type as specified in clause 21.6.2.

b) The **certs** component shall identify a list of updates to public-key certificate. It has the following subcomponents for each element:
   - The **subject** subcomponent shall hold the identification of the end-entity public-key certificate to be replaced.
   - The **serialNumber** subcomponent shall identify the end-entity public-key certificate or which new status information is available.
   - The **certStatus** shall hold the updated status information for the end-entity public-key certificate in question.

The authorizer shall verify the validity of the request by checking:

a) as specified in clause 21.4;

b) each element of the **certs** component for validity by checking–:
   - whether the **subject** subcomponent identifies a new entity and if not, return an **unknownSubject** error code;
   - whether the **serialNumber** subcomponent identifies a known public-key certificate and if not, return an **unknownCert** error code;

– whether the **certStatus** subcomponent has a valid value and if not, return an **unknownCertStatus** error code.

```
certUpdateRsp CONTENT-TYPE ::= {
                 CertUpdateRsp
  IDENTIFIED BY  id-certUpdateRsp }
```

The authorizer shall use the **certUpdateRsp** content type to report the outcome of the updates to status information on public-key certificates.

```
CertUpdateRsp ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  result        CHOICE {
    success       [0]  CertUpdateOK,
    failure       [1]  CertUpdateErr,
    ... },
  ... }

CertUpdateOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok          [0] SEQUENCE {
    subject       Name,
    serialNumber  CertificateSerialNumber,
    ... },
  not-ok      [1] SEQUENCE {
    status        CASP-CertStatusCode,
    ... },
  ... }

CertUpdateErr ::= SEQUENCE {
  code          CASP-error,
  ... }
```

The **CertUpdateRsp** data type specifies the actual content and has the following components:

The **CASPcommonComponents** data type as specified in clause 21.6.2.

The **result** component has the following two alternatives:

a) The **success** alternative shall be taken if the subscription was accepted for at least one public-key certificate. It shall then hold a value of the **CertUpdateOK** data type.

b) The **failure** alternative shall be taken if the evaluation of the request failed to a degree where no results could be returned. It shall then hold a value of the **CertUpdateErr** data type. The CASP-error data type is specified in clause 21.6.8.

The **CertUpdateOK** includes an element for each public-key certificate specified in the request in the same order. Each element has two alternatives:

a) The **ok** alternative shall be taken when the update to the public-key certificate information was successfully processed. It has the following components:

   – the **subject** component shall hold the name of the subject to which the public-key certificate had been issued;

   – the **serialNumber** component shall hold the serial number for the public-key certificate.

b) The **not-ok** alternative shall be taken when a corresponding public-key certificate was not identified.

   – the **no-reason** status code shall be returned when no code is applicable;

   – the **unknownCert** status code shall be selected when the corresponding element in the request did not identify a public-key certificate issued by the CA.

21.5.7 Certification authority subscription reject

```
rejectCAsubscribe  CONTENT-TYPE ::= {
                 RejectCAsubscribe
  IDENTIFIED BY  id-rejectCAsubscribe }
```

The **rejectCAsubscribe** content type is used by receiver of a response content to report problems with the response.

```
RejectCAsubscribe ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  reason        CASP-error,
  ... }
```

The **RejectCAsubscribe** data type specifies the actual content and has the following component:

The **sequence** component of the **CASPcommonComponents** data type shall take the same value as in the response on which it is reporting.

The **CASP-error** is specified in clause 21.6.8.

The authorizer shall verify the validity of a received response by checking

a) as specified in clause 21.4.

21.5.8 Certification authority subscription error codes

```
CASP-error ::= ENUMERATED {
  noReason                       (0),
  unknownContentType             (1),
  unsupportedCASPversion         (2),
  missingContent                 (3),
  missingContentComponent        (4),
  invalidContentComponent        (5),
  sequenceError                  (6),
  unknownCertStatus              (7),
  ... }
```

A value of the **CASP-error** data type indicates the result of an issued request.

a) the **noReason** value shall be selected when no other error code is applicable;

b) the **unknownContentType** value shall be selected if the content type is not known by the receiver;

c) the **unsupportedCASPversion** value shall be selected if a request or response content specified a CASP version that is not supported;

d) the **unsupportedContentVersion** value shall be selected when a request or response includes an unsupported content type;

e) the **missingContent** value shall be selected when the request or response did not include a content;

f) the **missingContentComponent** value shall be selected when a request or response does not includes a mandatory component;

g) the invalidContentComponent value shall be selected when an unexpected component was included in a request or response;

h) the **sequenceError** value shall be selected by when:

- an authorizer or a CA receives a request content for the first time that did not have the **sequence** component set to 1;
- an authorizer or a CA receives a request content that did not have the **sequence** component set to one higher than for a previous request content in the same direction; or
- an authorizer or a CA receives a response content with a **sequence** component value different from the one in the corresponding request content;

22 Trust broker protocol

The TB protocol allows the relying party to query the TB about the trustworthiness of the certificate issued to the subject it is communicating with. The relying party may send the certificate of either the issuing CA or the subject, and receive information about the trustworthiness of the subject's certificate.

There are two issues involved in determining the trustworthiness of a PKI certificate:

- Is the certificate still valid?
- How trustworthy is the CA that issued it?

The relying party may use a local client to answer the first question, in which case the TB will only answer the second one. Alternatively the relying party may ask the TB to answer both questions. The format of a TB request message is as follows:

```
TBrequest ::= CHOICE {
  caCert      [0] PKCertIdentifier,
  subjectCert [1] PKCertIdentifier,
  ... }
```

If the relying party sends the certificate of the CA, then only the second question above is to be answered. The TB does not know which of the CA's subjects the relying party is communicating with, and therefore can only return information that relates to all the certificates issued by this CA's certificate. This method protects the privacy of the relying party by not divulging the subjects that the relying party communicates with. The TB shall check that the CA's certificate is still valid and then return information about the trustworthiness of all the subject certificates issued by this CA.

If the relying party sends the certificate of the subject, then the TB shall check that the subject's certificate is still valid and then check that the CA's certificate is still valid and return information about the trustworthiness of all the subject certificates issued by this CA.

```
TBresponse ::= CHOICE {
  success [0]  TBOK,
  failure [1]  TBerror,
  ... }

TBOK ::= SEQUENCE {
  levelOfAssurance   [0]  INTEGER (0..100),
  confidenceLevel    [1]  INTEGER (0..100),
  validationTime     [2]  UTCTime,
  info                    UTF8String  OPTIONAL,
  ... }

TBerror ::= SEQUENCE {
  code        ENUMERATED {
    caCertInvalid         (1),
    unknownCert           (2),
    unknownCertStatus     (3),
    subjectCertRevoked    (4),
    incorrectCert         (5),
    contractExpired       (6),
    pathValidationFailed  (7),
    timeOut               (8),
    other                 (99),
    ... },
  diagnostic  UTF8String OPTIONAL,
  ... }
```

If the TB successfully validates the presented certificate, it returns the following information to the relying party:

levelOfAssurance -	indicates the level of assurance that the relying party can have in the subject's certificate, as determined by the responding TB. This assurance is asserted by the TB as a result of its careful analysis of many factors. Different TBs may therefore return different LoA values for the same presented certificate due to their different validation procedures. The LoA value will be in the range 0 to 100, where 0 indicates zero assurance and 100 indicates full assurance.
confidenceLevel -	indicates the confidence that the TB has in the LoA value that it has returned, where 0 indicates zero confidence and 100 indicates full confidence.
validationTime –	the date and time at which the TB last checked the revocation information for the presented certificate.
info -	contains an optional free form display string that the TB would like the client software to display to the user.

If the TB fails to validate the presented certificate, it returns an error code and diagnostic string. Note that if the TB validates the certificate but does not trust it or the issuer, it should return TBOK with a LoA of zero. The error code can be one of the following:

caCertInvalid –	either the presented CA certificate is not valid, or the certificate of a CA superior to the presented subject certificate is not valid
unknownCert –	the certificate submitted to the TB by the relying party is unknown and cannot be validated
unknownCertStatus –	the TB cannot determine the status of the (known) certificate submitted by the relying party
subjectCertRevoked –	the presented subject certificate has been revoked
incorrectCert –	the presented certificate is incorrectly formatted

contractExpired –	the relying party's contract with the TB has expired and should be renewed before service can be restored
pathValidationFailed –	one or more of the CA certificates in the chain from the subject's certificate to the root CA could not be validated. Another attempt at a later time may succeed
timeOut –	the presented certificate could not be validated because one or more services timed out. Retrying again later may rectify this error
other –	the presented certificate could not be validated for some reason other than one of the above. If this error code is used, then it is mandatory for the TB to complete the diagnostic string

The diagnostic string is an optional parameter that the TB may wish to be recorded in the client logs and displayed to the relying party. It is mandatory if the error code other is returned.

Annex A

Public-key and attribute certificate frameworks

(This annex forms an integral part of this Recommendation | International Standard.)

This annex includes all of the ASN.1 type, value, and information object class definitions contained in this Specification in the form of three ASN.1 modules: **`AuthenticationFramework`**, **`CertificateExtensions`** and **`AttributeCertificateDefinitions`**.

```
-- A.1 - Authentication framework module

AuthenticationFramework {joint-iso-itu-t ds(5) module(1) authenticationFramework(7) 8}
DEFINITIONS ::=
BEGIN

-- EXPORTS All
-- The types and values defined in this module are exported for use in the other ASN.1
-- modules contained within the Directory Specifications, and for the use of other
-- applications which will use them to access Directory services. Other applications may
-- use them for their own purposes, but this will not constrain extensions and
-- modifications needed to maintain or improve the Directory service.

IMPORTS
  basicAccessControl, certificateExtensions, id-asx, id-at, id-avr, id-ldx, id-lsx,
  id-mr, id-nf, id-oa, id-oc, id-sc, informationFramework, selectedAttributeTypes
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 8}

  ATTRIBUTE, DistinguishedName, MATCHING-RULE, Name, NAME-FORM, OBJECT-CLASS,
  RelativeDistinguishedName, SYNTAX-NAME, top
    FROM InformationFramework informationFramework

  bitStringMatch, boolean, booleanMatch, caseExactMatch, commonName,
  directoryString, distinguishedNameMatch, generalizedTime,
  generalizedTimeMatch, generalizedTimeOrderingMatch, integer, integerMatch,
  integerOrderingMatch, objectIdentifierMatch, octetString, octetStringMatch,
  UnboundedDirectoryString, UniqueIdentifier, uri
    FROM SelectedAttributeTypes selectedAttributeTypes

  algorithmIdentifierMatch, certificateExactMatch, certificateListExactMatch,
  certificatePairExactMatch, CertificatePoliciesSyntax, CertPolicyId, GeneralNames,
  KeyUsage,
  CertificateAssertion, CertificateExactAssertion, CertificateListAssertion,
  CertificateListExactAssertion, CertificatePairAssertion,
  CertificatePairExactAssertion
      FROM CertificateExtensions certificateExtensions ;


-- parameterized types

SIGNATURE ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}},
  signature            BIT STRING,
  ... }

SIGNED{ToBeSigned} ::= SEQUENCE {
  toBeSigned    ToBeSigned,
  COMPONENTS OF SIGNATURE,
  ... }

HASH{ToBeHashed} ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}},
  hashValue            BIT STRING (CONSTRAINED BY {
   -- shall be the result of applying a hashing procedure to the DER-encoded
   -- octets of a value of -- ToBeHashed } ),
  ... }

ENCRYPTED{ToBeEnciphered} ::= BIT STRING (CONSTRAINED BY {
    -- shall be the result of applying an encipherment procedure
    -- to the BER-encoded octets of a value of -- ToBeEnciphered } )

ENCRYPTED-HASH{ToBeSigned} ::= BIT STRING (CONSTRAINED BY {
  -- shall be the result of applying a hashing procedure to the DER-encoded (see 6.2)
  -- octets of a value of -- ToBeSigned -- and then applying an encipherment procedure
  -- to those octets -- } )

FINGERPRINT {ToBeFingerprinted} ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}},
  fingerprint          BIT STRING,
  ... }

ALGORITHM ::= CLASS {
  &Type           OPTIONAL,
  &id             OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
  [PARMS         &Type]
  IDENTIFIED BY &id }

AlgorithmIdentifier{ALGORITHM:SupportedAlgorithms} ::= SEQUENCE {
  algorithm   ALGORITHM.&id({SupportedAlgorithms}),
  parameters  ALGORITHM.&Type({SupportedAlgorithms}{@algorithm}) OPTIONAL,
  ... }

/* The definitions of the following information object set is deferred to referencing
specifications having a requirement for specific information object sets.*/

SupportedAlgorithms ALGORITHM ::= {...}

/* The definitions of the following information value set is deferred to referencing
specifications having a requirement for specific value sets.*/

SupportedCurves OBJECT IDENTIFIER ::= {dummyCurv, ...}

dummyCurv OBJECT IDENTIFIER ::= {2 5 5}

-- public-key certificate definition

Certificate ::= SIGNED{TBSCertificate}

TBSCertificate ::= SEQUENCE {
  version                  [0]  Version DEFAULT v1,
  serialNumber                  CertificateSerialNumber,
  signature                     AlgorithmIdentifier{{SupportedAlgorithms}},
  issuer                        Name,
  validity                      Validity,
  subject                       Name,
  subjectPublicKeyInfo          SubjectPublicKeyInfo,
  issuerUniqueIdentifier   [1] IMPLICIT UniqueIdentifier OPTIONAL,
  ...,
  [[2: -- if present, version shall be v2 or v3
  subjectUniqueIdentifier  [2] IMPLICIT UniqueIdentifier OPTIONAL]],
  [[3: -- if present, version shall be v2 or v3
  extensions               [3]  Extensions OPTIONAL]]
  -- If present, version shall be v3]]
 } (CONSTRAINED BY { -- shall be DER encoded -- } )

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

CertificateSerialNumber ::= INTEGER

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

SubjectPublicKeyInfo ::= SEQUENCE {
  algorithm         AlgorithmIdentifier{{SupportedAlgorithms}},
  subjectPublicKey  PublicKey,
  ... }
```

```
PublicKey ::= BIT STRING

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

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

-- For those extensions where ordering of individual extensions within the SEQUENCE is
-- significant, the specification of those individual extensions shall include the
-- rules for the significance of the order therein

Extension ::= SEQUENCE {
  extnId     EXTENSION.&id({ExtensionSet}),
  critical   BOOLEAN DEFAULT FALSE,
  extnValue  OCTET STRING
    (CONTAINING EXTENSION.&ExtnType({ExtensionSet}{@extnId})
       ENCODED BY der),
  ... }

der OBJECT IDENTIFIER ::=
  {joint-iso-itu-t asn1(1) ber-derived(2) distinguished-encoding(1)}

ExtensionSet EXTENSION ::= {...}

EXTENSION ::= CLASS {
  &id           OBJECT IDENTIFIER UNIQUE,
  &ExtnType }
WITH SYNTAX {
  SYNTAX        &ExtnType
  IDENTIFIED BY &id }

-- other PKI certificate constructs

Certificates ::= SEQUENCE {
  userCertificate    Certificate,
  certificationPath  ForwardCertificationPath OPTIONAL,
  ... }

ForwardCertificationPath ::= SEQUENCE SIZE (1..MAX) OF CrossCertificates

CrossCertificates ::= SET SIZE (1..MAX) OF Certificate

CertificationPath ::= SEQUENCE {
  userCertificate    Certificate,
  theCACertificates  SEQUENCE SIZE (1..MAX) OF CertificatePair OPTIONAL,
  ... }

PkiPath ::= SEQUENCE SIZE (1..MAX) OF Certificate

-- certificate revocation list (CRL)

CertificateList ::= SIGNED{CertificateListContent}

CertificateListContent ::= SEQUENCE {
  version               Version OPTIONAL,
  -- if present, version shall be v2
  signature             AlgorithmIdentifier{{SupportedAlgorithms}},
  issuer                Name,
  thisUpdate            Time,
  nextUpdate            Time OPTIONAL,
  revokedCertificates   SEQUENCE OF SEQUENCE {
    serialNumber          CertificateSerialNumber,
    revocationDate        Time,
    crlEntryExtensions    Extensions OPTIONAL,
    ...} OPTIONAL,
  ...,
  ...,
  crlExtensions    [0]  Extensions OPTIONAL }
```

```
CertAVL ::= SIGNED {TBSCertAVL}

TBSCertAVL ::= SEQUENCE {
  version                [0]  IMPLICIT Version DEFAULT v1,
  serialNumber                AvlSerialNumber OPTIONAL,
  signature                   AlgorithmIdentifier {{SupportedAlgorithms}},
  issuer                      Name,
  constrained                 BOOLEAN,
  entries                     SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    idType                      CHOICE {
      certIdentifier         [0]  PKCertIdentifier,
      entityGroup                 DistinguishedName, -- only for constrained = FALSE
      ... },
    scope                  [0]  IMPLICIT ScopeRestrictions OPTIONAL,
    entryExtensions        [1]  IMPLICIT Extensions OPTIONAL,
    ... },
  ...,
  ...,
  avlExtensions               Extensions OPTIONAL }

AvlSerialNumber ::= INTEGER (0..MAX)

PKCertIdentifier ::= CHOICE {
  issuerSerialNumber          IssuerSerialNumber,
  fingerprintPKC         [0]  IMPLICIT FINGERPRINT {Certificate},
  fingerprintPK          [1]  IMPLICIT FINGERPRINT {PublicKey},
  ... }

IssuerSerialNumber ::= SEQUENCE {
  issuer        Name,
  serialNumber  CertificateSerialNumber,
  ... }

ScopeRestrictions ::= SEQUENCE OF ScopeRestriction

SCOPE-RESTRICTION ::= TYPE-IDENTIFIER

ScopeRestriction ::= SEQUENCE {
  id            SCOPE-RESTRICTION.&id,
  restriction   SCOPE-RESTRICTION.&Type,
  ... }

protRestrict EXTENSION ::= {
  SYNTAX        ProtRestriction
  IDENTIFIED BY id-protRestrict }

ProtRestriction ::= SEQUENCE (SIZE (1..MAX)) OF OBJECT IDENTIFIER

-- PKI object classes

pkiUser OBJECT-CLASS ::= {
  SUBCLASS OF          {top}
  KIND                 auxiliary
  MAY CONTAIN          {userCertificate}
  LDAP-NAME            {"pkiUser"}
  LDAP-DESC            "X.509 PKI User"
  ID                   id-oc-pkiUser }

pkiCA OBJECT-CLASS ::= {
  SUBCLASS OF          {top}
  KIND                 auxiliary
  MAY CONTAIN          {cACertificate |
                        certificateRevocationList |
                        eepkCertificateRevocationList |
                        authorityRevocationList |
                        crossCertificatePair}
  LDAP-NAME            {"pkiCA"}
  LDAP-DESC            "X.509 PKI Certificate Authority"
  ID                   id-oc-pkiCA }

cRLDistributionPoint OBJECT-CLASS ::= {
  SUBCLASS OF          {top}
  KIND                 structural
  MUST CONTAIN         {commonName}
  MAY CONTAIN          {certificateRevocationList |
                        eepkCertificateRevocationList |
                        authorityRevocationList |
                        deltaRevocationList}
  LDAP-NAME            {"cRLDistributionPoint"}
  LDAP-DESC            "X.509 CRL distribution point"
  ID                   id-oc-cRLDistributionPoint }

cRLDistPtNameForm NAME-FORM ::= {
  NAMES                cRLDistributionPoint
  WITH ATTRIBUTES      {commonName}
  ID                   id-nf-cRLDistPtNameForm }

deltaCRL OBJECT-CLASS ::= {
  SUBCLASS OF          {top}
  KIND                 auxiliary
  MAY CONTAIN          {deltaRevocationList}
  LDAP-NAME            {"deltaCRL"}
  LDAP-DESC            "X.509 delta CRL"
  ID                   id-oc-deltaCRL }

cpCps OBJECT-CLASS ::= {
  SUBCLASS OF          {top}
  KIND                 auxiliary
  MAY CONTAIN          {certificatePolicy |
                        certificationPracticeStmt}
  LDAP-NAME            {"cpCps"}
  LDAP-DESC            "Certificate Policy and Certification Practice Statement"
  ID                   id-oc-cpCps }

pkiCertPath OBJECT-CLASS ::= {
  SUBCLASS OF          {top}
  KIND                 auxiliary
  MAY CONTAIN          {pkiPath}
  LDAP-NAME            {"pkiCertPath"}
  LDAP-DESC            "PKI Certification Path"
  ID                   id-oc-pkiCertPath }

-- PKI directory attributes

userCertificate ATTRIBUTE ::= {
  WITH SYNTAX               Certificate
  EQUALITY MATCHING RULE    certificateExactMatch
  LDAP-SYNTAX               x509Certificate.&id
  LDAP-NAME                 {"userCertificate"}
  LDAP-DESC                 "X.509 user certificate"
  ID                        id-at-userCertificate }

cACertificate ATTRIBUTE ::= {
  WITH SYNTAX               Certificate
  EQUALITY MATCHING RULE    certificateExactMatch
  LDAP-SYNTAX               x509Certificate.&id
  LDAP-NAME                 {"cACertificate"}
  LDAP-DESC                 "X.509 CA certificate"
  ID                        id-at-cAcertificate }

crossCertificatePair ATTRIBUTE ::= {
  WITH SYNTAX               CertificatePair
  EQUALITY MATCHING RULE    certificatePairExactMatch
  LDAP-SYNTAX               x509CertificatePair.&id
  LDAP-NAME                 {"crossCertificatePair"}
  LDAP-DESC                 "X.509 cross certificate pair"
  ID                        id-at-crossCertificatePair }

CertificatePair ::= SEQUENCE {
  issuedToThisCA   [0]  Certificate OPTIONAL,
  issuedByThisCA   [1]  Certificate OPTIONAL,
  ... }
```

```
   (WITH COMPONENTS { ..., issuedToThisCA PRESENT} |
    WITH COMPONENTS { ..., issuedByThisCA PRESENT})

certificateRevocationList ATTRIBUTE ::= {
  WITH SYNTAX              CertificateList
  EQUALITY MATCHING RULE   certificateListExactMatch
  LDAP-SYNTAX              x509CertificateList.&id
  LDAP-NAME                {"certificateRevocationList"}
  LDAP-DESC                "X.509 certificate revocation list"
  ID                       id-at-certificateRevocationList }

eepkCertificateRevocationList ATTRIBUTE ::= {
  WITH SYNTAX              CertificateList
  EQUALITY MATCHING RULE   certificateListExactMatch
  LDAP-SYNTAX              x509CertificateList.&id
  LDAP-NAME                {"certificateRevocationList"}
  LDAP-DESC                "X.509 EEPK certificate revocation list"
  ID                       id-at-eepkCertificateRevocationList }

authorityRevocationList ATTRIBUTE ::= {
  WITH SYNTAX              CertificateList
  EQUALITY MATCHING RULE   certificateListExactMatch
  LDAP-SYNTAX              x509CertificateList.&id
  LDAP-NAME                {"authorityRevocationList"}
  LDAP-DESC                "X.509 authority revocation list"
  ID                       id-at-authorityRevocationList }

deltaRevocationList ATTRIBUTE ::= {
  WITH SYNTAX              CertificateList
  EQUALITY MATCHING RULE   certificateListExactMatch
  LDAP-SYNTAX              x509CertificateList.&id
  LDAP-NAME                {"deltaRevocationList"}
  LDAP-DESC                "X.509 delta revocation list"
  ID                       id-at-deltaRevocationList }

supportedAlgorithms ATTRIBUTE ::= {
  WITH SYNTAX              SupportedAlgorithm
  EQUALITY MATCHING RULE   algorithmIdentifierMatch
  LDAP-SYNTAX              x509SupportedAlgorithm.&id
  LDAP-NAME                {"supportedAlgorithms"}
  LDAP-DESC                "X.509 support algorithms"
  ID                       id-at-supportedAlgorithms }

SupportedAlgorithm ::= SEQUENCE {
  algorithmIdentifier              AlgorithmIdentifier{{SupportedAlgorithms}},
  intendedUsage                [0] KeyUsage OPTIONAL,
  intendedCertificatePolicies  [1] CertificatePoliciesSyntax OPTIONAL,
  ... }

certificationPracticeStmt ATTRIBUTE ::= {
  WITH SYNTAX  InfoSyntax
  ID           id-at-certificationPracticeStmt }

InfoSyntax ::= CHOICE {
  content  UnboundedDirectoryString,
  pointer  SEQUENCE {
    name     GeneralNames,
    hash     HASH{HashedPolicyInfo} OPTIONAL,
    ... },
  ... }

POLICY ::= TYPE-IDENTIFIER

HashedPolicyInfo ::= POLICY.&Type({Policies})

Policies POLICY ::= {...} -- Defined by implementors

certificatePolicy ATTRIBUTE ::= {
  WITH SYNTAX  PolicySyntax
  ID           id-at-certificatePolicy }
```

```
PolicySyntax ::= SEQUENCE {
  policyIdentifier  PolicyID,
  policySyntax      InfoSyntax,
  ... }

PolicyID ::= CertPolicyId

pkiPath ATTRIBUTE ::= {
  WITH SYNTAX  PkiPath
  ID           id-at-pkiPath }

userPassword ATTRIBUTE ::= {
  WITH SYNTAX              OCTET STRING(SIZE (0..MAX))
  EQUALITY MATCHING RULE   octetStringMatch
  LDAP-SYNTAX              octetString.&id
  LDAP-NAME                {"userPassword"}
  ID                       id-at-userPassword }

-- LDAP syntaxes defined by IETF RFC 4523

x509Certificate SYNTAX-NAME ::= {
  LDAP-DESC         "X.509 Certificate"
  DIRECTORY SYNTAX  Certificate
  ID                id-lsx-x509Certificate }

x509CertificateList SYNTAX-NAME ::= {
  LDAP-DESC         "X.509 Certificate List"
  DIRECTORY SYNTAX  CertificateList
  ID                id-lsx-x509CertificateList }

x509CertificatePair SYNTAX-NAME ::= {
  LDAP-DESC         "X.509 Certificate Pair"
  DIRECTORY SYNTAX  CertificatePair
  ID                id-lsx-x509CertificatePair }

x509SupportedAlgorithm SYNTAX-NAME ::= {
  LDAP-DESC         "X.509 Supported Algorithm"
  DIRECTORY SYNTAX  SupportedAlgorithm
  ID                id-lsx-x509SupportedAlgorithm }

-- object identifier assignments

-- object classes

id-oc-cRLDistributionPoint          OBJECT IDENTIFIER ::= {id-oc 19}
id-oc-pkiUser                       OBJECT IDENTIFIER ::= {id-oc 21}
id-oc-pkiCA                         OBJECT IDENTIFIER ::= {id-oc 22}
id-oc-deltaCRL                      OBJECT IDENTIFIER ::= {id-oc 23}
id-oc-cpCps                         OBJECT IDENTIFIER ::= {id-oc 30}
id-oc-pkiCertPath                   OBJECT IDENTIFIER ::= {id-oc 31}

-- name forms

id-nf-cRLDistPtNameForm             OBJECT IDENTIFIER ::= {id-nf 14}

-- directory attributes

id-at-userPassword                  OBJECT IDENTIFIER ::= {id-at 35}
id-at-userCertificate               OBJECT IDENTIFIER ::= {id-at 36}
id-at-cAcertificate                 OBJECT IDENTIFIER ::= {id-at 37}
id-at-authorityRevocationList       OBJECT IDENTIFIER ::= {id-at 38}
id-at-certificateRevocationList     OBJECT IDENTIFIER ::= {id-at 39}
id-at-crossCertificatePair          OBJECT IDENTIFIER ::= {id-at 40}
id-at-supportedAlgorithms           OBJECT IDENTIFIER ::= {id-at 52}
id-at-deltaRevocationList           OBJECT IDENTIFIER ::= {id-at 53}
id-at-certificationPracticeStmt     OBJECT IDENTIFIER ::= {id-at 68}
id-at-certificatePolicy             OBJECT IDENTIFIER ::= {id-at 69}
id-at-pkiPath                       OBJECT IDENTIFIER ::= {id-at 70}
id-at-eepkCertificateRevocationList OBJECT IDENTIFIER ::= {id-at 101}

-- syntaxes
```

```
id-lsx-x509Certificate              OBJECT IDENTIFIER ::= {id-lsx 8}
id-lsx-x509CertificateList          OBJECT IDENTIFIER ::= {id-lsx 9}
id-lsx-x509CertificatePair          OBJECT IDENTIFIER ::= {id-lsx 10}
id-lsx-x509SupportedAlgorithm       OBJECT IDENTIFIER ::= {id-lsx 49}

-- Authorization validation restrictions

id-protRestrict                     OBJECT IDENTIFIER ::= {id-avr 1}

END -- AuthenticationFramework


-- A.2 - Certificate extensions module

CertificateExtensions {joint-iso-itu-t ds(5) module(1) certificateExtensions(26) 8}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS ALL

IMPORTS
  id-at, id-ce, id-ldx, id-mr, informationFramework, authenticationFramework,
  pkiPmiExternalDataTypes, selectedAttributeTypes
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 8}

  Name, RelativeDistinguishedName, Attribute{}, MATCHING-RULE,
  SupportedAttributes, SYNTAX-NAME
    FROM InformationFramework informationFramework

  AvlSerialNumber, CertificateSerialNumber, CertificateList, AlgorithmIdentifier{},
  EXTENSION, Time, PolicyID, SupportedAlgorithms
    FROM AuthenticationFramework authenticationFramework

  UnboundedDirectoryString
    FROM SelectedAttributeTypes selectedAttributeTypes

  ORAddress
    FROM PkiPmiExternalDataTypes pkiPmiExternalDataTypes;

-- Unless explicitly noted otherwise, there is no significance to the ordering
-- of components of a SEQUENCE OF construct in this Specification.

-- public-key certificate and CRL extensions

authorityKeyIdentifier EXTENSION ::= {
  SYNTAX         AuthorityKeyIdentifier
  IDENTIFIED BY  id-ce-authorityKeyIdentifier }

AuthorityKeyIdentifier ::= SEQUENCE {
  keyIdentifier             [0]  KeyIdentifier OPTIONAL,
  authorityCertIssuer       [1]  GeneralNames OPTIONAL,
  authorityCertSerialNumber [2]  CertificateSerialNumber OPTIONAL,
  ... }
  (WITH COMPONENTS {..., authorityCertIssuer        PRESENT,
                         authorityCertSerialNumber  PRESENT } |
   WITH COMPONENTS {..., authorityCertIssuer        ABSENT,
                         authorityCertSerialNumber  ABSENT } )

KeyIdentifier ::= OCTET STRING

subjectKeyIdentifier EXTENSION ::= {
  SYNTAX         SubjectKeyIdentifier
  IDENTIFIED BY  id-ce-subjectKeyIdentifier }

SubjectKeyIdentifier ::= KeyIdentifier

keyUsage EXTENSION ::= {
  SYNTAX         KeyUsage
  IDENTIFIED BY  id-ce-keyUsage }
```

```
KeyUsage ::= BIT STRING {
  digitalSignature  (0),
  contentCommitment (1),
  keyEncipherment   (2),
  dataEncipherment  (3),
  keyAgreement      (4),
  keyCertSign       (5),
  cRLSign           (6),
  encipherOnly      (7),
  decipherOnly      (8) }

extKeyUsage EXTENSION ::= {
  SYNTAX          SEQUENCE SIZE (1..MAX) OF KeyPurposeId
  IDENTIFIED BY   id-ce-extKeyUsage }

KeyPurposeId ::= OBJECT IDENTIFIER

privateKeyUsagePeriod EXTENSION ::= {
  SYNTAX          PrivateKeyUsagePeriod
  IDENTIFIED BY   id-ce-privateKeyUsagePeriod }

PrivateKeyUsagePeriod ::= SEQUENCE {
  notBefore   [0]  GeneralizedTime OPTIONAL,
  notAfter    [1]  GeneralizedTime OPTIONAL,
  ... }
  (WITH COMPONENTS {..., notBefore  PRESENT } |
   WITH COMPONENTS {..., notAfter   PRESENT } )

certificatePolicies EXTENSION ::= {
  SYNTAX          CertificatePoliciesSyntax
  IDENTIFIED BY   id-ce-certificatePolicies }

CertificatePoliciesSyntax ::= SEQUENCE SIZE (1..MAX) OF PolicyInformation

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

CertPolicyId ::= OBJECT IDENTIFIER

PolicyQualifierInfo ::= SEQUENCE {
  policyQualifierId  CERT-POLICY-QUALIFIER.&id({SupportedPolicyQualifiers}),
  qualifier          CERT-POLICY-QUALIFIER.&Qualifier
              ({SupportedPolicyQualifiers}{@policyQualifierId}) OPTIONAL,
  ... }

SupportedPolicyQualifiers CERT-POLICY-QUALIFIER ::= {...}

anyPolicy OBJECT IDENTIFIER ::= {id-ce-certificatePolicies 0}

CERT-POLICY-QUALIFIER ::= CLASS {
  &id                  OBJECT IDENTIFIER UNIQUE,
  &Qualifier           OPTIONAL }
WITH SYNTAX {
  POLICY-QUALIFIER-ID  &id
  [QUALIFIER-TYPE      &Qualifier] }

policyMappings EXTENSION ::= {
  SYNTAX          PolicyMappingsSyntax
  IDENTIFIED BY   id-ce-policyMappings }

PolicyMappingsSyntax ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
  issuerDomainPolicy   CertPolicyId,
  subjectDomainPolicy  CertPolicyId,
  ... }

authorizationValidation EXTENSION ::= {
  SYNTAX          AvlId
  IDENTIFIED BY   id-ce-authorizationValidation }
```

```
AvlId ::= SEQUENCE {
  issuer        Name,
  serialNumber  AvlSerialNumber OPTIONAL,
  ... }

subjectAltName EXTENSION ::= {
  SYNTAX         GeneralNames
  IDENTIFIED BY  id-ce-subjectAltName }

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

GeneralName ::= CHOICE {
  otherName                  [0]  INSTANCE OF OTHER-NAME,
  rfc822Name                 [1]  IA5String,
  dNSName                    [2]  IA5String,
  x400Address                [3]  ORAddress,
  directoryName              [4]  Name,
  ediPartyName               [5]  EDIPartyName,
  uniformResourceIdentifier  [6]  IA5String,
  iPAddress                  [7]  OCTET STRING,
  registeredID               [8]  OBJECT IDENTIFIER,
  ... }

OTHER-NAME ::= TYPE-IDENTIFIER

EDIPartyName ::= SEQUENCE {
  nameAssigner  [0]  UnboundedDirectoryString OPTIONAL,
  partyName     [1]  UnboundedDirectoryString,
  ... }

issuerAltName EXTENSION ::= {
  SYNTAX         GeneralNames
  IDENTIFIED BY  id-ce-issuerAltName }

subjectDirectoryAttributes EXTENSION ::= {
  SYNTAX         AttributesSyntax
  IDENTIFIED BY  id-ce-subjectDirectoryAttributes }

AttributesSyntax ::= SEQUENCE SIZE (1..MAX) OF Attribute{{SupportedAttributes}}

basicConstraints EXTENSION ::= {
  SYNTAX         BasicConstraintsSyntax
  IDENTIFIED BY  id-ce-basicConstraints }

BasicConstraintsSyntax ::= SEQUENCE {
  cA                 BOOLEAN DEFAULT FALSE,
  pathLenConstraint  INTEGER(0..MAX) OPTIONAL,
  ... }

nameConstraints EXTENSION ::= {
  SYNTAX         NameConstraintsSyntax
  IDENTIFIED BY  id-ce-nameConstraints }

NameConstraintsSyntax ::= SEQUENCE {
  permittedSubtrees   [0]  GeneralSubtrees OPTIONAL,
  excludedSubtrees    [1]  GeneralSubtrees OPTIONAL,
  ... }
  (WITH COMPONENTS {..., permittedSubtrees  PRESENT } |
   WITH COMPONENTS {..., excludedSubtrees   PRESENT } )

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 EXTENSION ::= {
  SYNTAX          PolicyConstraintsSyntax
  IDENTIFIED BY  id-ce-policyConstraints }

PolicyConstraintsSyntax ::= SEQUENCE {
  requireExplicitPolicy  [0]  SkipCerts OPTIONAL,
  inhibitPolicyMapping   [1]  SkipCerts OPTIONAL,
  ... }
  (WITH COMPONENTS {..., requireExplicitPolicy PRESENT } |
   WITH COMPONENTS {..., inhibitPolicyMapping  PRESENT } )

SkipCerts ::= INTEGER(0..MAX)

inhibitAnyPolicy EXTENSION ::= {
  SYNTAX          SkipCerts
  IDENTIFIED BY  id-ce-inhibitAnyPolicy }

cRLNumber EXTENSION ::= {
  SYNTAX          CRLNumber
  IDENTIFIED BY  id-ce-cRLNumber }

CRLNumber ::= INTEGER(0..MAX)

crlScope EXTENSION ::= {
  SYNTAX          CRLScopeSyntax
  IDENTIFIED BY  id-ce-cRLScope }

CRLScopeSyntax ::= SEQUENCE SIZE (1..MAX) OF PerAuthorityScope

PerAuthorityScope ::= SEQUENCE {
  authorityName        [0]  GeneralName OPTIONAL,
  distributionPoint    [1]  DistributionPointName OPTIONAL,
  onlyContains         [2]  OnlyCertificateTypes OPTIONAL,
  onlySomeReasons      [4]  ReasonFlags OPTIONAL,
  serialNumberRange    [5]  NumberRange OPTIONAL,
  subjectKeyIdRange    [6]  NumberRange OPTIONAL,
  nameSubtrees         [7]  GeneralNames OPTIONAL,
  baseRevocationInfo   [9]  BaseRevocationInfo OPTIONAL,
  ... }

OnlyCertificateTypes ::= BIT STRING {
  user      (0),
  authority (1),
  attribute (2)}

NumberRange ::= SEQUENCE {
  startingNumber  [0]  INTEGER OPTIONAL,
  endingNumber    [1]  INTEGER OPTIONAL,
  modulus              INTEGER OPTIONAL,
  ... }

BaseRevocationInfo ::= SEQUENCE {
  cRLStreamIdentifier    [0]  CRLStreamIdentifier OPTIONAL,
  cRLNumber              [1]  CRLNumber,
  baseThisUpdate         [2]  GeneralizedTime,
  ... }

statusReferrals EXTENSION ::= {
  SYNTAX          StatusReferrals
  IDENTIFIED BY  id-ce-statusReferrals }

StatusReferrals ::= SEQUENCE SIZE (1..MAX) OF StatusReferral

StatusReferral ::= CHOICE {
  cRLReferral     [0]  CRLReferral,
  otherReferral   [1]  INSTANCE OF OTHER-REFERRAL,
  ... }

CRLReferral ::= SEQUENCE {
  issuer          [0]  GeneralName OPTIONAL,
  location        [1]  GeneralName OPTIONAL,
  deltaRefInfo    [2]  DeltaRefInfo OPTIONAL,
  cRLScope              CRLScopeSyntax,
  lastUpdate       [3]  GeneralizedTime OPTIONAL,
  lastChangedCRL   [4]  GeneralizedTime OPTIONAL,
  ...
}

DeltaRefInfo ::= SEQUENCE {
  deltaLocation  GeneralName,
  lastDelta      GeneralizedTime OPTIONAL,
  ... }

OTHER-REFERRAL ::= TYPE-IDENTIFIER

cRLStreamIdentifier EXTENSION ::= {
  SYNTAX         CRLStreamIdentifier
  IDENTIFIED BY  id-ce-cRLStreamIdentifier }

CRLStreamIdentifier ::= INTEGER (0..MAX)

orderedList EXTENSION ::= {
  SYNTAX         OrderedListSyntax
  IDENTIFIED BY  id-ce-orderedList }

OrderedListSyntax ::= ENUMERATED {
  ascSerialNum (0),
  ascRevDate   (1),
  ...}

deltaInfo EXTENSION ::= {
  SYNTAX         DeltaInformation
  IDENTIFIED BY  id-ce-deltaInfo }

DeltaInformation ::= SEQUENCE {
  deltaLocation  GeneralName,
  nextDelta      GeneralizedTime OPTIONAL,
  ... }

toBeRevoked EXTENSION ::= {
  SYNTAX         ToBeRevokedSyntax
  IDENTIFIED BY  id-ce-toBeRevoked }

ToBeRevokedSyntax ::= SEQUENCE SIZE (1..MAX) OF ToBeRevokedGroup

ToBeRevokedGroup ::= SEQUENCE {
  certificateIssuer  [0]  GeneralName OPTIONAL,
  reasonInfo         [1]  ReasonInfo OPTIONAL,
  revocationTime          GeneralizedTime,
  certificateGroup        CertificateGroup,
  ... }

ReasonInfo ::= SEQUENCE {
  reasonCode            CRLReason,
  holdInstructionCode   HoldInstruction OPTIONAL,
  ... }

CertificateGroup ::= CHOICE {
  serialNumbers      [0]  CertificateSerialNumbers,
  serialNumberRange  [1]  CertificateGroupNumberRange,
  nameSubtree        [2]  GeneralName,
  ... }

CertificateGroupNumberRange ::= SEQUENCE {
  startingNumber  [0]  INTEGER,
  endingNumber    [1]  INTEGER,
  ... }

CertificateSerialNumbers ::= SEQUENCE SIZE (1..MAX) OF CertificateSerialNumber

revokedGroups EXTENSION ::= {
  SYNTAX         RevokedGroupsSyntax
  IDENTIFIED BY  id-ce-revokedGroups }
```

```
RevokedGroupsSyntax ::= SEQUENCE SIZE (1..MAX) OF RevokedGroup

RevokedGroup ::= SEQUENCE {
  certificateIssuer        [0]  GeneralName OPTIONAL,
  reasonInfo               [1]  ReasonInfo OPTIONAL,
  invalidityDate           [2]  GeneralizedTime OPTIONAL,
  revokedcertificateGroup  [3]  RevokedCertificateGroup,
  ... }

RevokedCertificateGroup ::= CHOICE {
  serialNumberRange  NumberRange,
  nameSubtree        GeneralName }

expiredCertsOnCRL EXTENSION ::= {
  SYNTAX         ExpiredCertsOnCRL
  IDENTIFIED BY  id-ce-expiredCertsOnCRL }

ExpiredCertsOnCRL ::= GeneralizedTime

reasonCode EXTENSION ::= {
  SYNTAX         CRLReason
  IDENTIFIED BY  id-ce-reasonCode }

CRLReason ::= ENUMERATED {
  unspecified          (0),
  keyCompromise        (1),
  cACompromise         (2),
  affiliationChanged   (3),
  superseded           (4),
  cessationOfOperation (5),
  certificateHold      (6),
  removeFromCRL        (8),
  privilegeWithdrawn   (9),
  aACompromise         (10),
  ...,
  weakAlgorithmOrKey   (11) }

holdInstructionCode EXTENSION ::= {
  SYNTAX         HoldInstruction
  IDENTIFIED BY  id-ce-holdInstructionCode }

HoldInstruction ::= OBJECT IDENTIFIER

invalidityDate EXTENSION ::= {
  SYNTAX         GeneralizedTime
  IDENTIFIED BY  id-ce-invalidityDate }

cRLDistributionPoints EXTENSION ::= {
  SYNTAX         CRLDistPointsSyntax
  IDENTIFIED BY  id-ce-cRLDistributionPoints }

CRLDistPointsSyntax ::= SEQUENCE SIZE (1..MAX) OF DistributionPoint

DistributionPoint ::= SEQUENCE {
  distributionPoint  [0]  DistributionPointName OPTIONAL,
  reasons            [1]  ReasonFlags OPTIONAL,
  cRLIssuer          [2]  GeneralNames OPTIONAL,
  ... }

DistributionPointName ::= CHOICE {
  fullName                 [0]  GeneralNames,
  nameRelativeToCRLIssuer  [1]  RelativeDistinguishedName,
  ... }

ReasonFlags ::= BIT STRING {
  unused                 (0),
  keyCompromise          (1),
  cACompromise           (2),
  affiliationChanged     (3),
  superseded             (4),
  cessationOfOperation   (5),
  certificateHold        (6),
  privilegeWithdrawn     (7),
  aACompromise           (8),
  weakAlgorithmOrKey     (9) }

issuingDistributionPoint EXTENSION ::= {
  SYNTAX         IssuingDistPointSyntax
  IDENTIFIED BY  id-ce-issuingDistributionPoint }

IssuingDistPointSyntax ::= SEQUENCE {
  -- If onlyContainsUserPublicKeyCerts and onlyContainsCACerts are both FALSE,
  -- the CRL covers both public-key certificate types
  distributionPoint              [0]  DistributionPointName OPTIONAL,
  onlyContainsUserPublicKeyCerts [1]  BOOLEAN DEFAULT FALSE,
  onlyContainsCACerts            [2]  BOOLEAN DEFAULT FALSE,
  onlySomeReasons                [3]  ReasonFlags OPTIONAL,
  indirectCRL                    [4]  BOOLEAN DEFAULT FALSE,
  onlyContainsAttributeCerts     [5]  BOOLEAN OPTIONAL, -- Use is strongly deprecated
  ... }

certificateIssuer EXTENSION ::= {
  SYNTAX         GeneralNames
  IDENTIFIED BY  id-ce-certificateIssuer }

deltaCRLIndicator EXTENSION ::= {
  SYNTAX         BaseCRLNumber
  IDENTIFIED BY  id-ce-deltaCRLIndicator }

BaseCRLNumber ::= CRLNumber

baseUpdateTime EXTENSION ::= {
  SYNTAX         GeneralizedTime
  IDENTIFIED BY  id-ce-baseUpdateTime }

freshestCRL EXTENSION ::= {
  SYNTAX         CRLDistPointsSyntax
  IDENTIFIED BY  id-ce-freshestCRL }

aAissuingDistributionPoint EXTENSION ::= {
  SYNTAX         AAIssuingDistPointSyntax
  IDENTIFIED BY  id-ce-aAissuingDistributionPoint }

AAIssuingDistPointSyntax ::= SEQUENCE {
  distributionPoint          [0]  DistributionPointName OPTIONAL,
  onlySomeReasons            [1]  ReasonFlags OPTIONAL,
  indirectCRL                [2]  BOOLEAN DEFAULT FALSE,
  containsUserAttributeCerts [3]  BOOLEAN DEFAULT TRUE,
  containsAACerts            [4]  BOOLEAN DEFAULT TRUE,
  containsSOAPublicKeyCerts  [5]  BOOLEAN DEFAULT TRUE,
  ... }

-- PKI matching rules

certificateExactMatch MATCHING-RULE ::= {
  SYNTAX       CertificateExactAssertion
  LDAP-SYNTAX  certExactAssertion.&id
  LDAP-NAME    {"certificateExactMatch"}
  LDAP-DESC    "X.509 Certificate Exact Match"
  ID           id-mr-certificateExactMatch }

CertificateExactAssertion ::= SEQUENCE {
  serialNumber  CertificateSerialNumber,
  issuer        Name,
  ... }

certificateMatch MATCHING-RULE ::= {
  SYNTAX       CertificateAssertion
  LDAP-SYNTAX  certAssertion.&id
  LDAP-NAME    {"certificateMatch"}
  LDAP-DESC    "X.509 Certificate Match"
  ID            id-mr-certificateMatch }

CertificateAssertion ::= SEQUENCE {
  serialNumber              [0]  CertificateSerialNumber OPTIONAL,
  issuer                    [1]  Name OPTIONAL,
  subjectKeyIdentifier      [2]  SubjectKeyIdentifier OPTIONAL,
  authorityKeyIdentifier    [3]  AuthorityKeyIdentifier OPTIONAL,
  certificateValid          [4]  Time OPTIONAL,
  privateKeyValid           [5]  GeneralizedTime OPTIONAL,
  subjectPublicKeyAlgID     [6]  OBJECT IDENTIFIER OPTIONAL,
  keyUsage                  [7]  KeyUsage OPTIONAL,
  subjectAltName            [8]  AltNameType OPTIONAL,
  policy                    [9]  CertPolicySet OPTIONAL,
  pathToName                [10] Name OPTIONAL,
  subject                   [11] Name OPTIONAL,
  nameConstraints           [12] NameConstraintsSyntax OPTIONAL,
  ... }

AltNameType ::= CHOICE {
  builtinNameForm  ENUMERATED {
    rfc822Name                   (1),
    dNSName                      (2),
    x400Address                  (3),
    directoryName                (4),
    ediPartyName                 (5),
    uniformResourceIdentifier    (6),
    iPAddress                    (7),
    registeredId                 (8),
    ...},
  otherNameForm    OBJECT IDENTIFIER,
  ... }

CertPolicySet ::= SEQUENCE SIZE (1..MAX) OF CertPolicyId

certificatePairExactMatch MATCHING-RULE ::= {
  SYNTAX        CertificatePairExactAssertion
  LDAP-SYNTAX   certPairExactAssertion.&id
  LDAP-NAME     {"certificatePairExactMatch"}
  LDAP-DESC     "X.509 Certificate Pair Exact Match"
  ID            id-mr-certificatePairExactMatch }

CertificatePairExactAssertion ::= SEQUENCE {
  issuedToThisCAAssertion  [0]  CertificateExactAssertion OPTIONAL,
  issuedByThisCAAssertion  [1]  CertificateExactAssertion OPTIONAL,
  ... }
  (WITH COMPONENTS { ..., issuedToThisCAAssertion  PRESENT } |
   WITH COMPONENTS { ..., issuedByThisCAAssertion  PRESENT } )

certificatePairMatch MATCHING-RULE ::= {
  SYNTAX        CertificatePairAssertion
  LDAP-SYNTAX   certPairAssertion.&id
  LDAP-NAME     {"certificatePairMatch"}
  LDAP-DESC     "X.509 Certificate Pair Match"
  ID            id-mr-certificatePairMatch }

CertificatePairAssertion ::= SEQUENCE {
  issuedToThisCAAssertion  [0]  CertificateAssertion OPTIONAL,
  issuedByThisCAAssertion  [1]  CertificateAssertion OPTIONAL,
  ... }
  (WITH COMPONENTS {..., issuedToThisCAAssertion  PRESENT } |
   WITH COMPONENTS {..., issuedByThisCAAssertion  PRESENT } )

certificateListExactMatch MATCHING-RULE ::= {
  SYNTAX        CertificateListExactAssertion
  LDAP-SYNTAX   certListExactAssertion.&id
  LDAP-NAME     {"certificateListExactMatch"}
  LDAP-DESC     "X.509 Certificate List Exact Match"
  ID            id-mr-certificateListExactMatch }

CertificateListExactAssertion ::= SEQUENCE {
  issuer                 Name,
  thisUpdate         Time,
  distributionPoint  DistributionPointName OPTIONAL }

certificateListMatch MATCHING-RULE ::= {
  SYNTAX  CertificateListAssertion
  LDAP-SYNTAX  certListAssertion.&id
  LDAP-NAME    {"certificateListMatch"}
  LDAP-DESC    "X.509 Certificate List Match"
  ID      id-mr-certificateListMatch }

CertificateListAssertion ::= SEQUENCE {
  issuer                          Name                     OPTIONAL,
  minCRLNumber               [0]  CRLNumber                OPTIONAL,
  maxCRLNumber               [1]  CRLNumber                OPTIONAL,
  reasonFlags                     ReasonFlags              OPTIONAL,
  dateAndTime                     Time                     OPTIONAL,
  distributionPoint          [2]  DistributionPointName    OPTIONAL,
  authorityKeyIdentifier     [3]  AuthorityKeyIdentifier OPTIONAL,
  ... }

algorithmIdentifierMatch MATCHING-RULE ::= {
  SYNTAX       AlgorithmIdentifier {{SupportedAlgorithms}}
  LDAP-SYNTAX  algorithmIdentifier.&id
  LDAP-NAME    {"algorithmIdentifierMatch"}
  LDAP-DESC    "X.509 Algorithm Identifier Match"
  ID           id-mr-algorithmIdentifierMatch }

policyMatch MATCHING-RULE ::= {
  SYNTAX  PolicyID
  ID      id-mr-policyMatch }

pkiPathMatch MATCHING-RULE ::= {
  SYNTAX  PkiPathMatchSyntax
  ID      id-mr-pkiPathMatch }

PkiPathMatchSyntax ::= SEQUENCE {
  firstIssuer  Name,
  lastSubject  Name,
  ... }

enhancedCertificateMatch MATCHING-RULE ::= {
  SYNTAX  EnhancedCertificateAssertion
  ID      id-mr-enhancedCertificateMatch }

EnhancedCertificateAssertion ::= SEQUENCE {
  serialNumber               [0]  CertificateSerialNumber OPTIONAL,
  issuer                     [1]  Name OPTIONAL,
  subjectKeyIdentifier       [2]  SubjectKeyIdentifier OPTIONAL,
  authorityKeyIdentifier     [3]  AuthorityKeyIdentifier OPTIONAL,
  certificateValid           [4]  Time OPTIONAL,
  privateKeyValid            [5]  GeneralizedTime OPTIONAL,
  subjectPublicKeyAlgID      [6]  OBJECT IDENTIFIER OPTIONAL,
  keyUsage                   [7]  KeyUsage OPTIONAL,
  subjectAltName             [8]  AltName OPTIONAL,
  policy                     [9]  CertPolicySet OPTIONAL,
  pathToName                 [10] GeneralNames OPTIONAL,
  subject                    [11] Name OPTIONAL,
  nameConstraints            [12] NameConstraintsSyntax OPTIONAL,
  ... }
  (ALL EXCEPT ({ -- none; at least one component shall be present --}))

AltName ::= SEQUENCE {
  altnameType   AltNameType,
  altNameValue  GeneralName OPTIONAL }

certExactAssertion SYNTAX-NAME ::= {
  LDAP-DESC         "X.509 Certificate Exact Assertion"
  DIRECTORY SYNTAX  CertificateExactAssertion
  ID                id-ldx-certExactAssertion }

certAssertion SYNTAX-NAME ::= {
  LDAP-DESC          "X.509 Certificate Assertion"
  DIRECTORY SYNTAX   CertificateAssertion
  ID                 id-ldx-certAssertion }

certPairExactAssertion SYNTAX-NAME ::= {
  LDAP-DESC          "X.509 Certificate Pair Exact Assertion"
  DIRECTORY SYNTAX   CertificatePairExactAssertion
  ID                 id-ldx-certPairExactAssertion }

certPairAssertion SYNTAX-NAME ::= {
  LDAP-DESC          "X.509 Certificate Pair Assertion"
  DIRECTORY SYNTAX   CertificatePairAssertion
  ID                 id-ldx-certPairAssertion }

certListExactAssertion SYNTAX-NAME ::= {
  LDAP-DESC          "X.509 Certificate List Exact Assertion"
  DIRECTORY SYNTAX   CertificateListExactAssertion
  ID                 id-ldx-certListExactAssertion }

certListAssertion SYNTAX-NAME ::= {
  LDAP-DESC          "X.509 Certificate List Assertion"
  DIRECTORY SYNTAX   CertificateListAssertion
  ID                 id-ldx-certListAssertion }

algorithmIdentifier SYNTAX-NAME ::= {
  LDAP-DESC          "X.509 Algorithm Identifier"
  DIRECTORY SYNTAX   AlgorithmIdentifier{{SupportedAlgorithms}}
  ID                 id-ldx-algorithmIdentifier }

-- Object identifier assignments

-- {id-ce 2} not used
-- {id-ce 3} not used
-- {id-ce 4} not used
-- {id-ce 5} not used
-- {id-ce 6} not used
-- {id-ce 7} not used
-- {id-ce 8} not used
id-ce-subjectDirectoryAttributes        OBJECT IDENTIFIER ::= {id-ce 9}
-- {id-ce 10} not used
-- {id-ce 11} not used
-- {id-ce 12} not used
-- {id-ce 13} not used
id-ce-subjectKeyIdentifier              OBJECT IDENTIFIER ::= {id-ce 14}
id-ce-keyUsage                          OBJECT IDENTIFIER ::= {id-ce 15}
id-ce-privateKeyUsagePeriod             OBJECT IDENTIFIER ::= {id-ce 16}
id-ce-subjectAltName                    OBJECT IDENTIFIER ::= {id-ce 17}
id-ce-issuerAltName                     OBJECT IDENTIFIER ::= {id-ce 18}
id-ce-basicConstraints                  OBJECT IDENTIFIER ::= {id-ce 19}
id-ce-cRLNumber                         OBJECT IDENTIFIER ::= {id-ce 20}
id-ce-reasonCode                        OBJECT IDENTIFIER ::= {id-ce 21}
-- {id-ce 22} not used
id-ce-holdInstructionCode               OBJECT IDENTIFIER ::= {id-ce 23}
id-ce-invalidityDate                    OBJECT IDENTIFIER ::= {id-ce 24}
-- {id-ce 25} not used
-- {id-ce 26} not used
id-ce-deltaCRLIndicator                 OBJECT IDENTIFIER ::= {id-ce 27}
id-ce-issuingDistributionPoint          OBJECT IDENTIFIER ::= {id-ce 28}
id-ce-certificateIssuer                 OBJECT IDENTIFIER ::= {id-ce 29}
id-ce-nameConstraints                   OBJECT IDENTIFIER ::= {id-ce 30}
id-ce-cRLDistributionPoints             OBJECT IDENTIFIER ::= {id-ce 31}
id-ce-certificatePolicies               OBJECT IDENTIFIER ::= {id-ce 32}
id-ce-policyMappings                    OBJECT IDENTIFIER ::= {id-ce 33}
-- deprecated                           OBJECT IDENTIFIER ::= {id-ce 34}
id-ce-authorityKeyIdentifier            OBJECT IDENTIFIER ::= {id-ce 35}
id-ce-policyConstraints                 OBJECT IDENTIFIER ::= {id-ce 36}
id-ce-extKeyUsage                       OBJECT IDENTIFIER ::= {id-ce 37}
-- id-ce-authorityAttributeIdentifier   OBJECT IDENTIFIER ::= {id-ce 38}
-- id-ce-roleSpecCertIdentifier         OBJECT IDENTIFIER ::= {id-ce 39}
id-ce-cRLStreamIdentifier               OBJECT IDENTIFIER ::= {id-ce 40}
-- id-ce-basicAttConstraints            OBJECT IDENTIFIER ::= {id-ce 41}
-- id-ce-delegatedNameConstraints        OBJECT IDENTIFIER ::= {id-ce 42}
-- id-ce-timeSpecification               OBJECT IDENTIFIER ::= {id-ce 43}
id-ce-cRLScope                           OBJECT IDENTIFIER ::= {id-ce 44}
id-ce-statusReferrals                    OBJECT IDENTIFIER ::= {id-ce 45}
id-ce-freshestCRL                        OBJECT IDENTIFIER ::= {id-ce 46}
id-ce-orderedList                        OBJECT IDENTIFIER ::= {id-ce 47}
-- id-ce-attributeDescriptor             OBJECT IDENTIFIER ::= {id-ce 48}
-- id-ce-userNotice                      OBJECT IDENTIFIER ::= {id-ce 49}
-- id-ce-sOAIdentifier                   OBJECT IDENTIFIER ::= {id-ce 50}
id-ce-baseUpdateTime                     OBJECT IDENTIFIER ::= {id-ce 51}
-- id-ce-acceptableCertPolicies          OBJECT IDENTIFIER ::= {id-ce 52}
id-ce-deltaInfo                          OBJECT IDENTIFIER ::= {id-ce 53}
id-ce-inhibitAnyPolicy                   OBJECT IDENTIFIER ::= {id-ce 54}
-- id-ce-targetingInformation            OBJECT IDENTIFIER ::= {id-ce 55}
-- id-ce-noRevAvail                      OBJECT IDENTIFIER ::= {id-ce 56}
-- id-ce-acceptablePrivilegePolicies     OBJECT IDENTIFIER ::= {id-ce 57}
id-ce-toBeRevoked                        OBJECT IDENTIFIER ::= {id-ce 58}
id-ce-revokedGroups                      OBJECT IDENTIFIER ::= {id-ce 59}
id-ce-expiredCertsOnCRL                  OBJECT IDENTIFIER ::= {id-ce 60}
-- id-ce-indirectIssuer                  OBJECT IDENTIFIER ::= {id-ce 61}
-- id-ce-noAssertion                     OBJECT IDENTIFIER ::= {id-ce 62}
id-ce-aAissuingDistributionPoint         OBJECT IDENTIFIER ::= {id-ce 63}
-- id-ce-issuedOnBehalfOf                OBJECT IDENTIFIER ::= {id-ce 64}
-- id-ce-singleUse                       OBJECT IDENTIFIER ::= {id-ce 65}
-- id-ce-groupAC                         OBJECT IDENTIFIER ::= {id-ce 66}
-- id-ce-allowedAttAss                   OBJECT IDENTIFIER ::= {id-ce 67}
-- id-ce-attributeMappings               OBJECT IDENTIFIER ::= {id-ce 68}
-- id-ce-holderNameConstraints           OBJECT IDENTIFIER ::= {id-ce 69}
id-ce-authorizationValidation            OBJECT IDENTIFIER ::= {id-ce 70}

-- matching rule OIDs

id-mr-certificateExactMatch       OBJECT IDENTIFIER ::= {id-mr 34}
id-mr-certificateMatch            OBJECT IDENTIFIER ::= {id-mr 35}
id-mr-certificatePairExactMatch   OBJECT IDENTIFIER ::= {id-mr 36}
id-mr-certificatePairMatch        OBJECT IDENTIFIER ::= {id-mr 37}
id-mr-certificateListExactMatch   OBJECT IDENTIFIER ::= {id-mr 38}
id-mr-certificateListMatch        OBJECT IDENTIFIER ::= {id-mr 39}
id-mr-algorithmIdentifierMatch    OBJECT IDENTIFIER ::= {id-mr 40}
id-mr-policyMatch                 OBJECT IDENTIFIER ::= {id-mr 60}
id-mr-pkiPathMatch                OBJECT IDENTIFIER ::= {id-mr 62}
id-mr-enhancedCertificateMatch    OBJECT IDENTIFIER ::= {id-mr 65}

-- Object identifiers for LDAP X.509 assertion syntaxes

id-ldx-certExactAssertion         OBJECT IDENTIFIER ::= {id-ldx 1}
id-ldx-certAssertion              OBJECT IDENTIFIER ::= {id-ldx 2}
id-ldx-certPairExactAssertion     OBJECT IDENTIFIER ::= {id-ldx 3}
id-ldx-certPairAssertion          OBJECT IDENTIFIER ::= {id-ldx 4}
id-ldx-certListExactAssertion     OBJECT IDENTIFIER ::= {id-ldx 5}
id-ldx-certListAssertion          OBJECT IDENTIFIER ::= {id-ldx 6}
id-ldx-algorithmIdentifier        OBJECT IDENTIFIER ::= {id-ldx 7}

END -- CertificateExtensions


--  A.3 - Attribute Certificate Framework module

AttributeCertificateDefinitions {joint-iso-itu-t ds(5) module(1)
  attributeCertificateDefinitions(32) 8}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS ALL

IMPORTS

  basicAccessControl, id-at, id-ce, id-mr, informationFramework,
  authenticationFramework, selectedAttributeTypes, id-oc, certificateExtensions,
  pkiPmiExternalDataTypes
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 8}
  ATTRIBUTE, Attribute{}, AttributeType, MATCHING-RULE, Name, OBJECT-CLASS,
  RelativeDistinguishedName, SupportedAttributes, SYNTAX-NAME, top
    FROM InformationFramework informationFramework

  AttributeTypeAndValue
    FROM BasicAccessControl basicAccessControl

  AlgorithmIdentifier, Certificate, CertificateList, CertificateSerialNumber,
    EXTENSION, Extensions, InfoSyntax, PolicySyntax, SIGNED{}, SupportedAlgorithms,
    x509CertificateList
    FROM AuthenticationFramework authenticationFramework

  TimeSpecification, UnboundedDirectoryString, UniqueIdentifier
    FROM SelectedAttributeTypes selectedAttributeTypes

  certificateListExactMatch, GeneralName, GeneralNames, NameConstraintsSyntax
    FROM CertificateExtensions certificateExtensions

  UserNotice
    FROM PkiPmiExternalDataTypes pkiPmiExternalDataTypes;

-- Unless explicitly noted otherwise, there is no significance to the ordering
-- of components of a SEQUENCE OF construct in this Specification.

-- attribute certificate constructs

AttributeCertificate ::= SIGNED{TBSAttributeCertificate}

TBSAttributeCertificate ::= SEQUENCE {
  version                 AttCertVersion, -- version is v2
  holder                  Holder,
  issuer                  AttCertIssuer,
  signature               AlgorithmIdentifier{{SupportedAlgorithms}},
  serialNumber            CertificateSerialNumber,
  attrCertValidityPeriod  AttCertValidityPeriod,
  attributes              SEQUENCE OF Attribute{{SupportedAttributes}},
  issuerUniqueID          UniqueIdentifier OPTIONAL,
  ...,
  ...,
  extensions              Extensions OPTIONAL
 }  (CONSTRAINED BY { -- shall be DER encoded -- } )

AttCertVersion ::= INTEGER {v2(1)}

Holder ::= SEQUENCE {
  baseCertificateID  [0]  IssuerSerial OPTIONAL,
  entityName         [1]  GeneralNames OPTIONAL,
  objectDigestInfo   [2]  ObjectDigestInfo OPTIONAL }
  (WITH COMPONENTS {..., baseCertificateID  PRESENT } |
   WITH COMPONENTS {..., entityName  PRESENT } |
   WITH COMPONENTS {..., objectDigestInfo  PRESENT } )

IssuerSerial ::= SEQUENCE {
  issuer     GeneralNames,
  serial     CertificateSerialNumber,
  issuerUID  UniqueIdentifier OPTIONAL,
  ... }

ObjectDigestInfo ::= SEQUENCE {
  digestedObjectType   ENUMERATED {
    publicKey         (0),
    publicKeyCert     (1),
    otherObjectTypes  (2)},
  otherObjectTypeID   OBJECT IDENTIFIER OPTIONAL,
  digestAlgorithm     AlgorithmIdentifier{{SupportedAlgorithms}},
  objectDigest        BIT STRING,
  ... }

AttCertIssuer ::= [0]  SEQUENCE {
  issuerName              GeneralNames OPTIONAL,
  baseCertificateID  [0]  IssuerSerial OPTIONAL,
  objectDigestInfo   [1]  ObjectDigestInfo OPTIONAL,
  ... }
  (WITH COMPONENTS {..., issuerName  PRESENT } |
   WITH COMPONENTS {..., baseCertificateID  PRESENT } |
   WITH COMPONENTS {..., objectDigestInfo  PRESENT } )

AttCertValidityPeriod ::= SEQUENCE {
  notBeforeTime  GeneralizedTime,
  notAfterTime   GeneralizedTime,
  ... }

AttributeCertificationPath ::= SEQUENCE {
  attributeCertificate  AttributeCertificate,
  acPath                SEQUENCE OF ACPathData OPTIONAL,
  ... }

ACPathData ::= SEQUENCE {
  certificate           [0]  Certificate OPTIONAL,
  attributeCertificate  [1]  AttributeCertificate OPTIONAL,
  ... }

PrivilegePolicy ::= OBJECT IDENTIFIER

-- privilege attributes

role ATTRIBUTE ::= {
  WITH SYNTAX  RoleSyntax
  ID           id-at-role }

RoleSyntax ::= SEQUENCE {
  roleAuthority  [0]  GeneralNames OPTIONAL,
  roleName       [1]  GeneralName,
  ... }

xmlPrivilegeInfo ATTRIBUTE ::= {
  WITH SYNTAX  UTF8String --contains XML-encoded privilege information
  ID           id-at-xMLPrivilegeInfo }

permission ATTRIBUTE ::= {
  WITH SYNTAX             DualStringSyntax
  EQUALITY MATCHING RULE  dualStringMatch
  ID                      id-at-permission }

DualStringSyntax ::= SEQUENCE {
  operation  [0]  UnboundedDirectoryString,
  object     [1]  UnboundedDirectoryString,
  ... }

dualStringMatch MATCHING-RULE ::= {
  SYNTAX  DualStringSyntax
  ID      id-mr-dualStringMatch }

timeSpecification EXTENSION ::= {
  SYNTAX         TimeSpecification
  IDENTIFIED BY  id-ce-timeSpecification }

timeSpecificationMatch MATCHING-RULE ::= {
  SYNTAX  TimeSpecification
  ID      id-mr-timeSpecMatch }

targetingInformation EXTENSION ::= {
  SYNTAX         SEQUENCE SIZE (1..MAX) OF Targets
  IDENTIFIED BY  id-ce-targetingInformation }

Targets ::= SEQUENCE SIZE (1..MAX) OF Target

Target ::= CHOICE {
  targetName   [0]  GeneralName,
  targetGroup  [1]  GeneralName,
  targetCert   [2]  TargetCert,
  ... }

TargetCert ::= SEQUENCE {
  targetCertificate  IssuerSerial,
  targetName         GeneralName OPTIONAL,
  certDigestInfo     ObjectDigestInfo OPTIONAL }

userNotice EXTENSION ::= {
  SYNTAX          SEQUENCE SIZE (1..MAX) OF UserNotice
  IDENTIFIED BY  id-ce-userNotice }

acceptablePrivilegePolicies EXTENSION ::= {
  SYNTAX          AcceptablePrivilegePoliciesSyntax
  IDENTIFIED BY  id-ce-acceptablePrivilegePolicies }

AcceptablePrivilegePoliciesSyntax ::= SEQUENCE SIZE (1..MAX) OF PrivilegePolicy

singleUse EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY  id-ce-singleUse }

groupAC EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY  id-ce-groupAC }

noRevAvail EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY  id-ce-noRevAvail }

sOAIdentifier EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY  id-ce-sOAIdentifier }

sOAIdentifierMatch MATCHING-RULE ::= {
  SYNTAX  NULL
  ID      id-mr-sOAIdentifierMatch }

attributeDescriptor EXTENSION ::= {
  SYNTAX          AttributeDescriptorSyntax
  IDENTIFIED BY  {id-ce-attributeDescriptor} }

AttributeDescriptorSyntax ::= SEQUENCE {
  identifier               AttributeIdentifier,
  attributeSyntax          OCTET STRING(SIZE (1..MAX)),
  name                [0]  AttributeName OPTIONAL,
  description         [1]  AttributeDescription OPTIONAL,
  dominationRule           PrivilegePolicyIdentifier,
  ... }

AttributeIdentifier ::= ATTRIBUTE.&id({AttributeIDs})

AttributeIDs ATTRIBUTE ::= {...}

AttributeName ::= UTF8String(SIZE (1..MAX))

AttributeDescription ::= UTF8String(SIZE (1..MAX))

PrivilegePolicyIdentifier ::= SEQUENCE {
  privilegePolicy  PrivilegePolicy,
  privPolSyntax    InfoSyntax,
  ... }

attDescriptor MATCHING-RULE ::= {
  SYNTAX  AttributeDescriptorSyntax
  ID      id-mr-attDescriptorMatch }

roleSpecCertIdentifier EXTENSION ::= {
  SYNTAX          RoleSpecCertIdentifierSyntax
  IDENTIFIED BY  {id-ce-roleSpecCertIdentifier} }

RoleSpecCertIdentifierSyntax ::=
  SEQUENCE SIZE (1..MAX) OF RoleSpecCertIdentifier

RoleSpecCertIdentifier ::= SEQUENCE {
  roleName              [0]  GeneralName,
  roleCertIssuer        [1]  GeneralName,
  roleCertSerialNumber  [2]  CertificateSerialNumber OPTIONAL,
  roleCertLocator       [3]  GeneralNames OPTIONAL,
  ... }

roleSpecCertIdMatch MATCHING-RULE ::= {
  SYNTAX  RoleSpecCertIdentifierSyntax
  ID      id-mr-roleSpecCertIdMatch }

basicAttConstraints EXTENSION ::= {
  SYNTAX         BasicAttConstraintsSyntax
  IDENTIFIED BY  {id-ce-basicAttConstraints} }

BasicAttConstraintsSyntax ::= SEQUENCE {
  authority          BOOLEAN DEFAULT FALSE,
  pathLenConstraint  INTEGER(0..MAX) OPTIONAL,
  ... }

basicAttConstraintsMatch MATCHING-RULE ::= {
  SYNTAX  BasicAttConstraintsSyntax
  ID      id-mr-basicAttConstraintsMatch }

delegatedNameConstraints EXTENSION ::= {
  SYNTAX         NameConstraintsSyntax
  IDENTIFIED BY  id-ce-delegatedNameConstraints }

delegatedNameConstraintsMatch MATCHING-RULE ::= {
  SYNTAX  NameConstraintsSyntax
  ID      id-mr-delegatedNameConstraintsMatch }

acceptableCertPolicies EXTENSION ::= {
  SYNTAX         AcceptableCertPoliciesSyntax
  IDENTIFIED BY  id-ce-acceptableCertPolicies }

AcceptableCertPoliciesSyntax ::= SEQUENCE SIZE (1..MAX) OF CertPolicyId

CertPolicyId ::= OBJECT IDENTIFIER

acceptableCertPoliciesMatch MATCHING-RULE ::= {
  SYNTAX  AcceptableCertPoliciesSyntax
  ID      id-mr-acceptableCertPoliciesMatch }

authorityAttributeIdentifier EXTENSION ::= {
  SYNTAX         AuthorityAttributeIdentifierSyntax
  IDENTIFIED BY  {id-ce-authorityAttributeIdentifier} }

AuthorityAttributeIdentifierSyntax ::= SEQUENCE SIZE (1..MAX) OF AuthAttId

AuthAttId ::= IssuerSerial

authAttIdMatch MATCHING-RULE ::= {
  SYNTAX  AuthorityAttributeIdentifierSyntax
  ID      id-mr-authAttIdMatch }

indirectIssuer EXTENSION ::= {
  SYNTAX         NULL
  IDENTIFIED BY  id-ce-indirectIssuer }

issuedOnBehalfOf EXTENSION ::= {
  SYNTAX         GeneralName
  IDENTIFIED BY  id-ce-issuedOnBehalfOf }

noAssertion EXTENSION ::= {
  SYNTAX         NULL
  IDENTIFIED BY  id-ce-noAssertion }

allowedAttributeAssignments EXTENSION ::= {
  SYNTAX         AllowedAttributeAssignments
  IDENTIFIED BY  id-ce-allowedAttributeAssignments }

AllowedAttributeAssignments ::= SET OF SEQUENCE {
  attributes              [0]  SET OF CHOICE {
    attributeType           [0]  AttributeType,
    attributeTypeandValues  [1]  Attribute{{SupportedAttributes}},
    ... },
  holderDomain            [1]  GeneralName,
  ... }

attributeMappings EXTENSION ::= {
  SYNTAX         AttributeMappings
  IDENTIFIED BY  id-ce-attributeMappings }

AttributeMappings ::= SET OF CHOICE {
  typeMappings      [0]  SEQUENCE {
    local             [0]  AttributeType,
    remote            [1]  AttributeType,
    ... },
  typeValueMappings [1]  SEQUENCE {
    local             [0]  AttributeTypeAndValue,
    remote            [1]  AttributeTypeAndValue,
    ... } }

holderNameConstraints EXTENSION ::= {
  SYNTAX         HolderNameConstraintsSyntax
  IDENTIFIED BY  id-ce-holderNameConstraints }

HolderNameConstraintsSyntax ::= SEQUENCE {
  permittedSubtrees  [0]  GeneralSubtrees,
  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)

-- PMI object classes

pmiUser OBJECT-CLASS ::= {
  SUBCLASS OF   {top}
  KIND          auxiliary
  MAY CONTAIN   {attributeCertificateAttribute}
  ID            id-oc-pmiUser }

pmiAA OBJECT-CLASS ::= { -- a PMI AA
  SUBCLASS OF   {top}
  KIND          auxiliary
  MAY CONTAIN   {aACertificate |
                 attributeCertificateRevocationList |
                 attributeAuthorityRevocationList}
  ID            id-oc-pmiAA }

pmiSOA OBJECT-CLASS ::= { -- a PMI Source of Authority
  SUBCLASS OF   {top}
  KIND          auxiliary
  MAY CONTAIN   {attributeCertificateRevocationList |
                 attributeAuthorityRevocationList |
                 attributeDescriptorCertificate}
  ID            id-oc-pmiSOA }

attCertCRLDistributionPt OBJECT-CLASS ::= {
  SUBCLASS OF   {top}
  KIND          auxiliary
  MAY CONTAIN  {attributeCertificateRevocationList |
                attributeAuthorityRevocationList}
  ID           id-oc-attCertCRLDistributionPts }

pmiDelegationPath OBJECT-CLASS ::= {
  SUBCLASS OF  {top}
  KIND         auxiliary
  MAY CONTAIN  {delegationPath}
  ID           id-oc-pmiDelegationPath }

privilegePolicy OBJECT-CLASS ::= {
  SUBCLASS OF  {top}
  KIND         auxiliary
  MAY CONTAIN  {privPolicy}
  ID           id-oc-privilegePolicy }

protectedPrivilegePolicy OBJECT-CLASS ::= {
  SUBCLASS OF  {top}
  KIND         auxiliary
  MAY CONTAIN  {protPrivPolicy}
  ID           id-oc-protectedPrivilegePolicy }

-- PMI directory attributes

attributeCertificateAttribute ATTRIBUTE ::= {
  WITH SYNTAX             AttributeCertificate
  EQUALITY MATCHING RULE  attributeCertificateExactMatch
  ID                      id-at-attributeCertificate }

aACertificate ATTRIBUTE ::= {
  WITH SYNTAX             AttributeCertificate
  EQUALITY MATCHING RULE  attributeCertificateExactMatch
  ID                      id-at-aACertificate }

attributeDescriptorCertificate ATTRIBUTE ::= {
  WITH SYNTAX             AttributeCertificate
  EQUALITY MATCHING RULE  attributeCertificateExactMatch
  ID                      id-at-attributeDescriptorCertificate }

attributeCertificateRevocationList ATTRIBUTE ::= {
  WITH SYNTAX             CertificateList
  EQUALITY MATCHING RULE  certificateListExactMatch
  LDAP-SYNTAX             x509CertificateList.&id
  LDAP-NAME               {"AttrCertificateRevocationList"}
  LDAP-DESC               "X.509 Attr certificate revocation list"
  ID                      id-at-attributeCertificateRevocationList }

eeAttrCertificateRevocationList ATTRIBUTE ::= {
  WITH SYNTAX             CertificateList
  EQUALITY MATCHING RULE  certificateListExactMatch
  LDAP-SYNTAX             x509CertificateList.&id
  LDAP-NAME               {"EEAttrCertificateRevocationList"}
  LDAP-DESC               "X.509 EEAttr certificate revocation list"
  ID                      id-at-eeAttrCertificateRevocationList }

attributeAuthorityRevocationList ATTRIBUTE ::= {
  WITH SYNTAX             CertificateList
  EQUALITY MATCHING RULE  certificateListExactMatch
  LDAP-SYNTAX             x509CertificateList.&id
  LDAP-NAME               {"AACertificateRevocationList"}
  LDAP-DESC               "X.509 AA certificate revocation list"
  ID                      id-at-attributeAuthorityRevocationList }

delegationPath ATTRIBUTE ::= {
  WITH SYNTAX  AttCertPath
  ID           id-at-delegationPath }

AttCertPath ::= SEQUENCE OF AttributeCertificate

privPolicy ATTRIBUTE ::= {
  WITH SYNTAX  PolicySyntax
  ID           id-at-privPolicy }

protPrivPolicy ATTRIBUTE ::= {
  WITH SYNTAX              AttributeCertificate
  EQUALITY MATCHING RULE   attributeCertificateExactMatch
  ID                       id-at-protPrivPolicy }

xmlPrivPolicy ATTRIBUTE ::= {
  WITH SYNTAX  UTF8String -- XML-encoded privilege policy information
  ID           id-at-xmlPrivPolicy }

-- Attribute certificate extensions and matching rules

attributeCertificateExactMatch MATCHING-RULE ::= {
  SYNTAX  AttributeCertificateExactAssertion
  ID      id-mr-attributeCertificateExactMatch }

AttributeCertificateExactAssertion ::= SEQUENCE {
  serialNumber  CertificateSerialNumber,
  issuer        AttCertIssuer,
  ... }

attributeCertificateMatch MATCHING-RULE ::= {
  SYNTAX  AttributeCertificateAssertion
  ID      id-mr-attributeCertificateMatch }

AttributeCertificateAssertion ::= SEQUENCE {
  holder              [0]  CHOICE {
    baseCertificateID  [0]  IssuerSerial,
    holderName         [1]  GeneralNames,
    ...} OPTIONAL,
  issuer              [1]  GeneralNames OPTIONAL,
  attCertValidity     [2]  GeneralizedTime OPTIONAL,
  attType             [3]  SET OF AttributeType OPTIONAL,
  ... }

-- At least one component of the sequence shall be present

holderIssuerMatch MATCHING-RULE ::= {
  SYNTAX  HolderIssuerAssertion
  ID      id-mr-holderIssuerMatch }

HolderIssuerAssertion ::= SEQUENCE {
  holder  [0]  Holder OPTIONAL,
  issuer  [1]  AttCertIssuer OPTIONAL,
  ... }

delegationPathMatch MATCHING-RULE ::= {
  SYNTAX  DelMatchSyntax
  ID      id-mr-delegationPathMatch }

DelMatchSyntax ::= SEQUENCE {
  firstIssuer  AttCertIssuer,
  lastHolder   Holder,
  ... }

extensionPresenceMatch MATCHING-RULE ::= {
  SYNTAX  EXTENSION.&id
  ID      id-mr-extensionPresenceMatch }

-- object identifier assignments

-- object classes

id-oc-pmiUser                         OBJECT IDENTIFIER ::= {id-oc 24}
id-oc-pmiAA                           OBJECT IDENTIFIER ::= {id-oc 25}
id-oc-pmiSOA                          OBJECT IDENTIFIER ::= {id-oc 26}
id-oc-attCertCRLDistributionPts       OBJECT IDENTIFIER ::= {id-oc 27}
id-oc-privilegePolicy                 OBJECT IDENTIFIER ::= {id-oc 32}
id-oc-pmiDelegationPath               OBJECT IDENTIFIER ::= {id-oc 33}
id-oc-protectedPrivilegePolicy        OBJECT IDENTIFIER ::= {id-oc 34}
```

```
-- directory attributes

id-at-attributeCertificate                 OBJECT IDENTIFIER ::= {id-at 58}
id-at-attributeCertificateRevocationList OBJECT IDENTIFIER ::= {id-at 59}
id-at-aACertificate                        OBJECT IDENTIFIER ::= {id-at 61}
id-at-attributeDescriptorCertificate      OBJECT IDENTIFIER ::= {id-at 62}
id-at-attributeAuthorityRevocationList   OBJECT IDENTIFIER ::= {id-at 63}
id-at-privPolicy                           OBJECT IDENTIFIER ::= {id-at 71}
id-at-role                                 OBJECT IDENTIFIER ::= {id-at 72}
id-at-delegationPath                       OBJECT IDENTIFIER ::= {id-at 73}
id-at-protPrivPolicy                       OBJECT IDENTIFIER ::= {id-at 74}
id-at-xMLPrivilegeInfo                     OBJECT IDENTIFIER ::= {id-at 75}
id-at-xmlPrivPolicy                        OBJECT IDENTIFIER ::= {id-at 76}
id-at-permission                           OBJECT IDENTIFIER ::= {id-at 82}
id-at-eeAttrCertificateRevocationList     OBJECT IDENTIFIER ::= {id-at 102}

-- attribute certificate extensions

id-ce-authorityAttributeIdentifier        OBJECT IDENTIFIER ::= {id-ce 38}
id-ce-roleSpecCertIdentifier               OBJECT IDENTIFIER ::= {id-ce 39}
id-ce-basicAttConstraints                  OBJECT IDENTIFIER ::= {id-ce 41}
id-ce-delegatedNameConstraints             OBJECT IDENTIFIER ::= {id-ce 42}
id-ce-timeSpecification                    OBJECT IDENTIFIER ::= {id-ce 43}
id-ce-attributeDescriptor                  OBJECT IDENTIFIER ::= {id-ce 48}
id-ce-userNotice                           OBJECT IDENTIFIER ::= {id-ce 49}
id-ce-sOAIdentifier                        OBJECT IDENTIFIER ::= {id-ce 50}
id-ce-acceptableCertPolicies               OBJECT IDENTIFIER ::= {id-ce 52}
id-ce-targetingInformation                 OBJECT IDENTIFIER ::= {id-ce 55}
id-ce-noRevAvail                           OBJECT IDENTIFIER ::= {id-ce 56}
id-ce-acceptablePrivilegePolicies          OBJECT IDENTIFIER ::= {id-ce 57}
id-ce-indirectIssuer                       OBJECT IDENTIFIER ::= {id-ce 61}
id-ce-noAssertion                          OBJECT IDENTIFIER ::= {id-ce 62}
id-ce-issuedOnBehalfOf                     OBJECT IDENTIFIER ::= {id-ce 64}
id-ce-singleUse                            OBJECT IDENTIFIER ::= {id-ce 65}
id-ce-groupAC                              OBJECT IDENTIFIER ::= {id-ce 66}
id-ce-allowedAttributeAssignments          OBJECT IDENTIFIER ::= {id-ce 67}
id-ce-attributeMappings                    OBJECT IDENTIFIER ::= {id-ce 68}
id-ce-holderNameConstraints                OBJECT IDENTIFIER ::= {id-ce 69}

-- PMI matching rules

id-mr-attributeCertificateMatch            OBJECT IDENTIFIER ::= {id-mr 42}
id-mr-attributeCertificateExactMatch       OBJECT IDENTIFIER ::= {id-mr 45}
id-mr-holderIssuerMatch                    OBJECT IDENTIFIER ::= {id-mr 46}
id-mr-authAttIdMatch                       OBJECT IDENTIFIER ::= {id-mr 53}
id-mr-roleSpecCertIdMatch                  OBJECT IDENTIFIER ::= {id-mr 54}
id-mr-basicAttConstraintsMatch             OBJECT IDENTIFIER ::= {id-mr 55}
id-mr-delegatedNameConstraintsMatch        OBJECT IDENTIFIER ::= {id-mr 56}
id-mr-timeSpecMatch                        OBJECT IDENTIFIER ::= {id-mr 57}
id-mr-attDescriptorMatch                   OBJECT IDENTIFIER ::= {id-mr 58}
id-mr-acceptableCertPoliciesMatch          OBJECT IDENTIFIER ::= {id-mr 59}
id-mr-delegationPathMatch                  OBJECT IDENTIFIER ::= {id-mr 61}
id-mr-sOAIdentifierMatch                   OBJECT IDENTIFIER ::= {id-mr 66}
id-mr-extensionPresenceMatch               OBJECT IDENTIFIER ::= {id-mr 67}
id-mr-dualStringMatch                      OBJECT IDENTIFIER ::= {id-mr 69}

END -- AttributeCertificateDefinitions

-- A.4 - PKI Protocol specifications module

PkiPmiWrapper {joint-iso-itu-t ds(5) module(1) pkiPmiWrapper(42) 8}
DEFINITIONS ::=
BEGIN

-- EXPORTS All

IMPORTS

  -- from Rec. ITU-T X.501 | ISO/IEC 9594-2
  attributeCertificateDefinitions, authenticationFramework, certificateExtensions, id-
cmsct, informationFramework, selectedAttributeTypes
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 8}

  Attribute{}, ATTRIBUTE, Name
    FROM InformationFramework informationFramework

  -- from Rec. ITU-T X.509 | ISO/IEC 9594-8

  ALGORITHM, AlgorithmIdentifier{}, Certificate, CertificateList, CertificateSerialNumber,
CertAVL,
  ENCRYPTED-HASH{}, PKCertIdentifier, SIGNATURE{},   TBSCertAVL,
  Version, AvlSerialNumber, PkiPath, SIGNED
    FROM AuthenticationFramework authenticationFramework

  CRLReason, SubjectKeyIdentifier
    FROM CertificateExtensions certificateExtensions

  AttributeCertificate
    FROM AttributeCertificateDefinitions attributeCertificateDefinitions

  -- from Rec. ITU-T X.520 | ISO/IEC 9594-6

  objectIdentifierMatch, octetStringMatch
    FROM SelectedAttributeTypes selectedAttributeTypes ;

WRAPPED-PDU ::= TYPE-IDENTIFIER

PDU-wrapper ::= SIGNED{TBSPDU-wrapper}

TBSPDU-wrapper ::= SEQUENCE  {
  version                   Version DEFAULT v1,
  signatureAlgorithm        AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
  certPath          [0]   IMPLICIT PkiPath,
  signedAttrs       [1]   IMPLICIT SignedAttributes OPTIONAL,
  conf                      CHOICE {
    clear             [2]   WrappedPDUInfo,
    protected         [3]   EncryptedInfo,
   ... },
  ... }

SupportedSignatureAlgorithms ALGORITHM ::= {...}

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

SupportedSignedAttributes ATTRIBUTE ::= { contentType | messageDigest }

WrappedPDUInfo ::= SEQUENCE {
  pduType       WRAPPED-PDU.&id ({SupportedPduSet}),
  pduInfo       WRAPPED-PDU.&Type ({SupportedPduSet}{@pduType}),
  ... }

SupportedPduSet WRAPPED-PDU ::= {...}

EncryptedInfo ::= SEQUENCE {
  keyAgreement      KeyAgreement,
  encryptedPduInfo  EncryptedPduInfo,
  ... }

KeyAgreement ::= SEQUENCE {
  senderDhInfo        [0] SenderDhInfo,
  keyEncryptionAlgorithm SEQUENCE {
    algorithm    ALGORITHM.&id ({SupportedKeyEncryptionAlgorithm}),
    parameters   ALGORITHM.&Type({SupportedKeyEncryptionAlgorithm}{@.algorithm}),
    ... },
  ... }

SupportedKeyEncryptionAlgorithm ALGORITHM ::= {...}

SenderDhInfo ::= CHOICE {
  senderStaticInfo    [0] SenderStaticInfo,
  senderDhPublicKey  [1] SenderDhPublicKey,
  ... }

SenderStaticInfo::= SEQUENCE {
  issuer       Name,
  serialNumber CertificateSerialNumber,
  partyAinfo   UserKeyingMaterial,
  ... }

SenderDhPublicKey ::= SEQUENCE {
  algorithm   AlgorithmIdentifier {{SupportedDHPublicKeyAlgorithms}},
  publicKey   BIT STRING,
  ... }

SupportedDHPublicKeyAlgorithms ALGORITHM ::= {...}

UserKeyingMaterial ::= OCTET STRING (SIZE (64))

EncryptedPduInfo ::= SEQUENCE {
  pduType                  WRAPPED-PDU.&id ({SupportedPduSet}),
  encryptedKey             EncryptedKey OPTIONAL,
  pduEncryptionAlgorithm   SEQUENCE {
    algorithm                ALGORITHM.&id ({SymmetricEncryptionAlgorithms}),
    parameter                ALGORITHM.&Type
                ({SymmetricEncryptionAlgorithms}{@.algorithm})} OPTIONAL,
  encryptedPdu          [0] EncryptedPdu,
  ... }

EncryptedKey ::= OCTET STRING

SymmetricEncryptionAlgorithms ALGORITHM ::= {...}

EncryptedPdu ::= OCTET STRING

SupportedAttributes ATTRIBUTE ::= {...}

AttributeCertificateV2 ::= AttributeCertificate

-- Attribute type specification as defined by IETF RFC 5652

contentType ATTRIBUTE ::= {
  WITH SYNTAX             WRAPPED-PDU.&id({SupportedPduSet})
  EQUALITY MATCHING RULE objectIdentifierMatch
  SINGLE VALUE            TRUE
  ID                      id-contentType }

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

messageDigest ATTRIBUTE ::= {
  WITH SYNTAX             OCTET STRING
  EQUALITY MATCHING RULE octetStringMatch
  SINGLE VALUE            TRUE
  ID                      id-messageDigest }

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

PkiWaError ::= ENUMERATED {
  unsupportedWrapperVersion          (0),
  unsupportedSignatureAlgorithm      (1),
  incompleteCertPath                 (2),
  certificationPathFailure           (3),
  invalidSignature                   (4),
  missingMandatoryAttributes         (5),
  unwantedAttribute                  (6),
  unsupportedPduType                 (7),
  unexpectedPduType                  (8),
  invalidPduSyntax                   (9),
  unknownDHpkCetificate              (10),
  invalidKeyingMaterial              (11),
  dhAlgorithmMismatch                  (12),
  invalideDhPublickey                  (13),
  unsupportedKeyWrappingAlgorithm      (14),
  keyEncAlgorithmParametersMissing     (15),
  keyEncAlgorithmParametersNotAllowed  (16),
  invalidParmsForSymEncryptAlgorithms  (17),
  decryptionFailed                     (18),
  ... }

END -- PkiPmiWrapper

-- A.5 - PKI-PMI protocol specifications

PkiPMIProtocolSpecifications {joint-iso-itu-t ds(5) module(1)
    pkiPMIProtocolSpecifications(43) 8}
DEFINITIONS ::=
BEGIN

-- EXPORTS All

IMPORTS

  -- from Rec. ITU-T X.501 | ISO/IEC 9594-2

  attributeCertificateDefinitions, authenticationFramework, certificateExtensions,
  id-cmsct, informationFramework, pkiPmiWrapper, selectedAttributeTypes
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 8}

  Attribute{}, ATTRIBUTE, Name, SupportedAttributes
    FROM InformationFramework informationFramework

  -- from Rec. ITU-T X.509 | ISO/IEC 9594-8

  ALGORITHM, AlgorithmIdentifier{}, Certificate, CertificateList, CertificateSerialNumber,
CertAVL,
  ENCRYPTED-HASH{}, PKCertIdentifier, SIGNATURE{},   TBSCertAVL,
  Version, AvlSerialNumber, PkiPath
    FROM AuthenticationFramework authenticationFramework

  CRLReason, SubjectKeyIdentifier
    FROM CertificateExtensions certificateExtensions

  AttributeCertificate
    FROM AttributeCertificateDefinitions attributeCertificateDefinitions

  PkiWaError, WRAPPED-PDU
    FROM PkiPmiWrapper pkiPmiWrapper

  -- from Rec. ITU-T X.520 | ISO/IEC 9594-6

  objectIdentifierMatch, octetStringMatch
    FROM SelectedAttributeTypes selectedAttributeTypes ;

-- PDU types

AvlPduSet WRAPPED-PDU ::= {
  certReq |
  certRsp |
  addAvlReq |
  addAvlRsp |
  replaceAvlReq |
  replaceAvlRsp |
  deleteAvlReq |
  deleteAvlRsp |
  rejectAVL |
  certSubscribeReq |
  certSubscribeRsp |
  certUnsubscribeReq |
  certUnsubscribeRsp |
  certReplaceReq |
  certReplaceRsp |
  rejectCAsubscribe,
  ... }

-- Authorization validation list management

AVMPcommonComponents ::= SEQUENCE {
  version    AVMPversion DEFAULT v1,
  timeStamp  GeneralizedTime,
  sequence   AVMPsequence,
  ... }

AVMPversion ::= ENUMERATED { v1(1), v2(2), v3(3), ... }

AVMPsequence ::= INTEGER (1..MAX)

certReq WRAPPED-PDU ::= {
                CertReq
  IDENTIFIED BY id-certReq }

CertReq ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  ... }

certRsp WRAPPED-PDU ::= {
                CertRsp
  IDENTIFIED BY id-certRsp }

CertRsp ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  result        CHOICE {
    success       [0]  CertOK,
    failure       [1]  CertErr,
    ... },
  ... }

CertOK ::= SEQUENCE {
  dhCert  Certificate,
  ... }

CertErr ::= SEQUENCE {
  notOK  CHOICE {
    wrErr    [0]  PkiWaError,
    avmpErr  [1]  AVMP-error,
    ... },
  note   Notifications OPTIONAL,
  ... }

Notifications ::= SEQUENCE SIZE (1..MAX) OF Attribute {{SupportedAttributes}}

addAvlReq WRAPPED-PDU ::= {
                AddAvlReq
  IDENTIFIED BY id-addAvlReq }

AddAvlReq ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  certlist      CertAVL,
  ... }

addAvlRsp WRAPPED-PDU ::= {
                 AddAvlRsp
  IDENTIFIED BY  id-addAvlRsp }

AddAvlRsp ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  result        CHOICE {
    success       [0]  AddAvlOK,
    failure       [1]  AddAvlErr,
    ... },
  ... }

AddAvlOK ::= SEQUENCE {
  ok      NULL,
  ... }

AddAvlErr ::= SEQUENCE {
  notOK   AVMP-error,
  ... }

replaceAvlReq WRAPPED-PDU ::= {
                   ReplaceAvlReq
  IDENTIFIED BY  id-replaceAvlReq }

ReplaceAvlReq ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  old           AvlSerialNumber OPTIONAL,
  new           CertAVL,
  ... }

replaceAvlRsp WRAPPED-PDU ::= {
                   ReplaceAvlRsp
  IDENTIFIED BY  id-replaceAvlRsp }

ReplaceAvlRsp ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  result        CHOICE {
    success       [0]  RepAvlOK,
    failure       [1]  RepAvlErr,
    ... },
  ... }

RepAvlOK ::= SEQUENCE {
  ok      NULL,
  ... }

RepAvlErr ::= SEQUENCE {
  notOK   AVMP-error,
  ... }

deleteAvlReq WRAPPED-PDU ::= {
                   DeleteAvlReq
  IDENTIFIED BY  id-deleteAvlReq }

DeleteAvlReq ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  avl-Id        AvlSerialNumber OPTIONAL,
  ... }

deleteAvlRsp WRAPPED-PDU ::= {
                   DeleteAvlRsp
  IDENTIFIED BY  id-deleteAvlRsp }

DeleteAvlRsp ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  result        CHOICE {
    success       [0]  DelAvlOK,
    failure       [1]  DelAvlErr,
    ... },
  ... }

DelAvlOK ::= SEQUENCE {
  ok      NULL,
  ... }

DelAvlErr ::= SEQUENCE {
  notOK   AVMP-error,
  ... }

rejectAVL  WRAPPED-PDU ::= {
                   RejectAVL
  IDENTIFIED BY  id-rejectAVL }

RejectAVL ::= SEQUENCE {
  COMPONENTS OF AVMPcommonComponents,
  reason        AVMP-error,
  ... }

-- CA subscription

CASPcommonComponents ::= SEQUENCE {
  version    CASPversion DEFAULT v1,
  sequence   CASPsequence,
  ... }

CASPversion ::= ENUMERATED { v1(1), v2(2), v3(3), ... }

CASPsequence ::= INTEGER (1..MAX)

certSubscribeReq WRAPPED-PDU ::= {
                 CertSubscribeReq
  IDENTIFIED BY  id-certSubscribeReq }

CertSubscribeReq ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  certs   SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  ... }

certSubscribeRsp WRAPPED-PDU ::= {
                 CertSubscribeRsp
  IDENTIFIED BY  id-certSubscribeRsp }

CertSubscribeRsp ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  result       CHOICE {
    success        [0]  CertSubscribeOK,
    failure        [1]  CertSubscribeErr,
    ... },
  ... }

CertSubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok         [0] SEQUENCE {
    cert         Certificate,
    status       CertStatus,
    revokeReason CRLReason OPTIONAL,
    ... },
  not-ok     [1] SEQUENCE {
    status       CASP-CertStatusCode,
    ... },
  ... }

CertStatus ::= ENUMERATED {
  good     (0),
  revoked  (1),
  on-hold  (2),
  expired  (3),
  ... }

CASP-CertStatusCode ::= ENUMERATED {
  noReason       (1),
  unknownCert    (2),
  ... }

CertSubscribeErr ::= SEQUENCE {
  code       CASP-error,
  ... }

certUnsubscribeReq WRAPPED-PDU ::= {
                 CertUnsubscribeReq
  IDENTIFIED BY  id-certUnsubscribeReq }

CertUnsubscribeReq ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  certs  SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  ... }

certUnsubscribeRsp WRAPPED-PDU ::= {
                 CertUnsubscribeRsp
  IDENTIFIED BY  id-certUnsubscribeRsp }

CertUnsubscribeRsp ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  result       CHOICE {
    success        [0]  CertUnsubscribeOK,
    failure        [1]  CertUnsubscribeErr,
    ... },
  ... }

CertUnsubscribeOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok        [0] SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    ... },
  not-ok    [1] SEQUENCE {
    status       CASP-CertStatusCode,
    ... },
  ... }

CertUnsubscribeErr ::= SEQUENCE {
  code         CASP-error,
  ... }

certReplaceReq WRAPPED-PDU ::= {
                 CertReplaceReq
  IDENTIFIED BY  id-certReplaceReq }

CertReplaceReq ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  certs          SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    old            CertificateSerialNumber,
    new            Certificate,
    ... },
  ... }

certReplaceRsp WRAPPED-PDU ::= {
                 CertReplaceRsp
  IDENTIFIED BY  id-certReplaceRsp }

CertReplaceRsp ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  result         CHOICE {
    success        [0]  CertReplaceOK,
    failure        [1]  CertReplaceErr,
    ... },
  ... }

CertReplaceOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok         [0] SEQUENCE {
    issuer         Name,
    serialNumber   CertificateSerialNumber,
    ... },
  not-ok     [1] SEQUENCE {
    status         CASP-CertStatusCode,
    ... },
  ... }

CertReplaceErr ::= SEQUENCE {
  code         CHOICE {
    signedData      [0]  SignedData-error,
    envelopedData   [1]  EnvelopedData-error,
    casp           [2]  CASP-error,
    ... },
  ... }

certUpdateReq WRAPPED-PDU ::= {
                 CertUpdateReq
  IDENTIFIED BY  id-certUpdateReq }

CertUpdateReq ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  certs  SEQUENCE (SIZE (1..MAX)) OF SEQUENCE {
    subject      Name,
    serialNumber CertificateSerialNumber,
    certStatus   CertStatus,
    ... },
  ... }

certUpdateRsp WRAPPED-PDU ::= {
                 CertUpdateRsp
  IDENTIFIED BY  id-certUpdateRsp }

CertUpdateRsp ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  result        CHOICE {
    success       [0]  CertUpdateOK,
    failure       [1]  CertUpdateErr,
    ... },
  ... }

CertUpdateOK ::= SEQUENCE (SIZE (1..MAX)) OF CHOICE {
  ok         [0] SEQUENCE {
    subject       Name,
    serialNumber  CertificateSerialNumber,
    ... },
  not-ok     [1] SEQUENCE {
    status        CASP-CertStatusCode,
    ... },
  ... }

CertUpdateErr ::= SEQUENCE {
  code          CASP-error,
  ... }

rejectCAsubscribe  WRAPPED-PDU ::= {
                 RejectCAsubscribe
  IDENTIFIED BY  id-rejectCAsubscribe }

RejectCAsubscribe ::= SEQUENCE {
  COMPONENTS OF CASPcommonComponents,
  reason        CASP-error,
  ... }

SignedData-error ::= ENUMERATED {
  noReason                            (0),
  signedDataContectTypeExpected       (1),
  wrongSignedDataVersion              (2),
  missingContent                      (3),
  missingContentComponent             (4),
  invalidContentComponent             (5),
  unsupportedHashAlgorithm            (6),
  ... }

EnvelopedData-error ::= ENUMERATED {
  noReason                            (0),
  ... }

AVMP-error ::= ENUMERATED {
  noReason                            (0),
  unknownAvlEntity                    (1),
  unknownContentType                  (2),
  unsupportedAVMPversion              (3),
  missingContent                     (4),
  missingContentComponent            (5),
  invalidContentComponent            (6),
  sequenceError                      (7),
  protocolError                      (8),
  invalidAvlSignature                (9),
  duplicateAVL                       (10),
  missingAvlComponent                (11),
  invalidAvlVersion                  (12),
  notAllowedForConstrainedAVLEntity  (13),
  constrainedRequired                (14),
  nonConstrainedRequired             (15),
  unsupportedCriticalEntryExtension  (16),
  unsupportedCriticalExtension       (17),
  maxAVLsExceeded                    (18),
  unknownCert                        (19),
  unknownAVL                         (20),
  unsupportedScopeRestriction        (21),
  ... }

CASP-error ::= ENUMERATED {
  noReason                      (0),
  unknownContentType            (1),
  unsupportedWLMPversion        (2),
  missingContent                (3),
  missingContentComponent       (4),
  invalidContentComponent       (5),
  sequenceError                 (6),
  unknownSubject                (7),
  unknownCert                   (8),
  ... }

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

id-envelopedData OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840)
rsadsi(113549) pkcs(1) pkcs7(7) 3}

id-certReq               OBJECT IDENTIFIER ::= {id-cmsct 0}
id-certRsp               OBJECT IDENTIFIER ::= {id-cmsct 1}
id-addAvlReq             OBJECT IDENTIFIER ::= {id-cmsct 2}
id-addAvlRsp             OBJECT IDENTIFIER ::= {id-cmsct 3}
id-replaceAvlReq         OBJECT IDENTIFIER ::= {id-cmsct 4}
id-replaceAvlRsp         OBJECT IDENTIFIER ::= {id-cmsct 5}
id-updateAvlReq          OBJECT IDENTIFIER ::= {id-cmsct 6}
id-updateAvlRsp          OBJECT IDENTIFIER ::= {id-cmsct 7}
id-deleteAvlReq          OBJECT IDENTIFIER ::= {id-cmsct 8}
id-deleteAvlRsp          OBJECT IDENTIFIER ::= {id-cmsct 9}
id-rejectAVL             OBJECT IDENTIFIER ::= {id-cmsct 10}
id-certSubscribeReq      OBJECT IDENTIFIER ::= {id-cmsct 11}
id-certSubscribeRsp      OBJECT IDENTIFIER ::= {id-cmsct 12}
id-certUnsubscribeReq    OBJECT IDENTIFIER ::= {id-cmsct 13}
id-certUnsubscribeRsp    OBJECT IDENTIFIER ::= {id-cmsct 14}
id-certReplaceReq        OBJECT IDENTIFIER ::= {id-cmsct 15}
id-certReplaceRsp        OBJECT IDENTIFIER ::= {id-cmsct 16}
id-certUpdateReq         OBJECT IDENTIFIER ::= {id-cmsct 17}
id-certUpdateRsp         OBJECT IDENTIFIER ::= {id-cmsct 18}
id-rejectCAsubscribe     OBJECT IDENTIFIER ::= {id-cmsct 19}


-- Trust broker protocol

TBrequest ::= CHOICE {
  caCert      [0] PKCertIdentifier,
  subjectCert [1] PKCertIdentifier,
  ... }

TBresponse ::= CHOICE {
  success [0]  TBOK,
  failure [1]  TBerror,
  ... }
```

```
TBOK ::= SEQUENCE {
  levelOfAssurance  [0]  INTEGER (0..100),
  confidenceLevel   [1]  INTEGER (0..100),
  validationTime    [2]  UTCTime,
  info                   UTF8String  OPTIONAL,
  ... }

TBerror ::= SEQUENCE {
  code        ENUMERATED {
    caCertInvalid        (1),
    unknownCert          (2),
    unknownCertStatus    (3),
    subjectCertRevoked   (4),
    incorrectCert        (5),
    contractExpired      (6),
    pathValidationFailed (7),
    timeOut              (8),
    other                (99),
    ... },
  diagnostic  UTF8String OPTIONAL,
  ... }

END -- PkiPMIProtocolSpecifications
```

Annex B

Reference definition of cryptographic algorithms

(This annex does not form an integral part of this Recommendation | International Standard.)

This annex lists cryptographic algorithms defined in other specifications, by firstly listing the allocated object identifiers for the cryptographic algorithms followed by the actual algorithm definitions. The definitions take the form of the ASN.1 module, **`AlgorithmObjectIdentifiers`**.

```
AlgorithmObjectIdentifiers {joint-iso-itu-t ds(5) module(1)
  algorithmObjectIdentifiers(8) 8}
DEFINITIONS ::=
BEGIN

-- EXPORTS All

/*
The values defined in this module are primarily taking from various specifications and
collected here for easy reference by other specifcations.

Wen values are copied form an IETF RFC, the IETF RFC number is shown.

When values are copied from the NIST Computer Security Objects Register (CSOR),
the label CSOR is used.
*/

IMPORTS
  algorithm, authenticationFramework
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 8}

  ALGORITHM, AlgorithmIdentifier{}, SupportedAlgorithms, SupportedCurves
    FROM AuthenticationFramework authenticationFramework;

ID ::= OBJECT IDENTIFIER

-- Object identifier allocations

-- Object identifiers allocated by this Specification (but not used)

nullAlgorithm            ID ::= {algorithm 0}
encryptionAlgorithm      ID ::= {algorithm 1}
hashAlgorithm            ID ::= {algorithm 2}
signatureAlgorithm       ID ::= {algorithm 3}

-- synonyms

id-ea                    ID ::= encryptionAlgorithm
id-ha                    ID ::= hashAlgorithm
id-sa                    ID ::= signatureAlgorithm

-- the following object identifier assignments reserve values assigned to deprecated
functions

id-ea-rsa                ID ::= {id-ea 1}
id-ha-sqMod-n            ID ::= {id-ha 1}
id-sa-sqMod-nWithRSA     ID ::= {id-sa 1}

-- object identifiers allocated by other organization

us-iso                   ID ::= { iso(1) member-body(2) us(840) }
ansi-x9-57               ID ::= { us-iso ansi-x9-57(10040) }
ansi-x9-62               ID ::= { us-iso ansi-x962(10045) }
ansi-x9-42               ID ::= { us-iso ansi-x942(10046) }
iso-standard             ID ::= { iso(1) standard(0) }
iso9797                  ID ::= { iso-standard message-authentication-codes(9797) }
iso-organization         ID ::= { iso(1) identified-organization(3) }
certicom                 ID ::= { iso-organization certicom(132) }
certicom-curve           ID ::= { certicom curve(0) }
```

```
teletrust                ID ::= { iso-organization teletrust(36) }
ecStdCurvesAndGen        ID ::= { teletrust algorithm(3) signaturealgorithm(3) ecSign(2) 8}
versionOne               ID ::= { ecStdCurvesAndGen ellipticCurve(1) versionOne(1) }

us-joint                 ID ::= { joint-iso-itu-t(2) country(16) us(840) }
usgov                    ID ::= { us-joint organization(1) gov(101) }
dodAlgorithms            ID ::= { usgov dod(2) infosec(1) algorithms(1) }
csor                     ID ::= { usgov csor(3) }
nistAlgorithms           ID ::= { csor nistAlgorithm(4) } -- CSOR
aes                      ID ::= { nistAlgorithms 1 } -- CSOR
hashAlgs                 ID ::= { nistAlgorithms 2 } -- CSOR
sigAlgs                  ID ::= { nistAlgorithms 3 } -- CSOR

rsadsi                   ID ::= { iso(1) member-body(2) us(840) rsadsi(113549) }
pkcs-1                   ID ::= { rsadsi pkcs(1) pkcs-1(1) }
digestAlgorithm          ID ::= { rsadsi digestAlgorithm(2) }


-- Symmetric key algorithm object identifiers

id-aes128-CBC            ID ::= { aes 2 }  -- CSOR
id-aes192-CBC            ID ::= { aes 22 } -- CSOR
id-aes256-CBC            ID ::= { aes 42 } -- CSOR

-- AES key wrap algorithms from IETF RFC 3394

id-aes128-wrap           ID ::= { aes 5 }
id-aes192-wrap           ID ::= { aes 25 }
id-aes256-wrap           ID ::= { aes 45 }

-- Pubkic key algorithm object identifiers

rsaEncryption            ID ::= { pkcs-1 rsaEncryption(1)} -- IETF RFC 4055
id-keyExchangeAlgorithm ID ::= { dodAlgorithms id-keyExchangeAlgorithm(22)}
                                  -- IETF RFC 3279
id-dsa                   ID ::= { ansi-x9-57 x9algorithm(4) 1 }    -- IETF RFC 5480
id-ecPublicKey           ID ::= { ansi-x9-62 keyType(2) 1 }        -- IETF RFC 5480
id-ecDH                  ID ::= { certicom schemes(1) ecdh(12) }  -- IETF RFC 5480
id-ecMQV                 ID ::= { certicom schemes(1) ecmqv(13) } -- IETF RFC 5480
dh-public-number         ID ::= { ansi-x9-42 number-type(2) dh-public-number(1) } --IETF
RFC 2631


-- Hash algorithms object identifiers

-- The OID for SHA hash algorithms are specified in NIST FIPS PUB 180-4

id-sha1                  ID ::= {iso(1) identified-organization(3) oiw(14) secsig(3)
                                 algorithms(2) 26} -- IETF RFC 3279
id-sha256                ID ::= { hashAlgs 1 } -- CSOR
id-sha384                ID ::= { hashAlgs 2 } -- CSOR
id-sha512                ID ::= { hashAlgs 3 } -- CSOR
id-sha224                ID ::= { hashAlgs 4 } -- CSOR
id-sha512-224            ID ::= { hashAlgs 5 } -- CSOR
id-sha512-256            ID ::= { hashAlgs 6 } -- CSOR

hashAlg                  ID ::= {  iso(1) identified-organization(3) dod(6) internet(1)
                                   private(4) enterprise(1) kudelski(1722)
                                   cryptography(12) 2 } -- BLAKE2, RFC 7693

-- SIGNATURE ALGORITHM IDS

-- RSASSA-PKCS1-v1_5 signature algorithm object identifiers (From IETF RFC 3447)

sha1WithRSAEncryption    ID ::= { pkcs-1 sha1WithRSAEncryption(5) }
sha256WithRSAEncryption ID ::= { pkcs-1 sha256WithRSAEncryption(11) }
sha384WithRSAEncryption ID ::= { pkcs-1 sha384WithRSAEncryption(12) }
sha512WithRSAEncryption ID ::= { pkcs-1 sha512WithRSAEncryption(13) }
sha224WithRSAEncryption ID ::= { pkcs-1 sha224WithRSAEncryption(14) }

-- RSASSA-PSS signature algorithm object identifiers (From IETF RFC 4055)
```

```
id-RSASSA-PSS           ID ::= { pkcs-1 10 }
id-mgf1                 ID ::= { pkcs-1 8 }

-- DSA algorithms object identifiers

id-dsa-with-sha1        ID ::= {iso(1) member-body(2) us(840) x9-57(10040) x9algorithm(4)
                                dsa-with-sha1(3)}
id-dsa-with-sha224      ID ::= { sigAlgs 1 } -- CSOR
id-dsa-with-sha256      ID ::= { sigAlgs 2 } -- CSOR

-- From IETF RFC 5758
ecdsa-with-SHA224       ID ::= { ansi-x9-62 signatures(4)
                                            ecdsa-with-SHA2(3) 1 }
ecdsa-with-SHA256       ID ::= { ansi-x9-62 signatures(4)
                                            ecdsa-with-SHA2(3) 2 }
ecdsa-with-SHA384       ID ::= { ansi-x9-62 signatures(4)
                                            ecdsa-with-SHA2(3) 3 }
ecdsa-with-SHA512       ID ::= { ansi-x9-62 signatures(4) ecdsa-with-SHA2(3) 4 }

--  Object identifier for curves

-- From IETF RFC 5480

secp192r1       ID ::= { ansi-x9-62 curves(3) prime(1) 1 }
sect163k1       ID ::= { certicom-curve 1 }
sect163r2       ID ::= { certicom-curve 15 }
secp224r1       ID ::= { certicom-curve 33 }
sect233k1       ID ::= { certicom-curve 26 }
sect233r1       ID ::= { certicom-curve 27 }
secp256r1       ID ::= { ansi-x9-62 curves(3) prime(1) 7 }
sect283k1       ID ::= { certicom-curve 16 }
sect283r1       ID ::= { certicom-curve 17 }
secp384r1       ID ::= { certicom-curve 34 }
sect409k1       ID ::= { certicom-curve 36 }
sect409r1       ID ::= { certicom-curve 37 }
secp521r1       ID ::= { certicom-curve 35 }
sect571k1       ID ::= { certicom-curve 38 }
sect571r1       ID ::= { certicom-curve 39 }

-- From IETF RFC 5639

brainpoolP160r1 ID ::= { versionOne 1 }
brainpoolP160t1 ID ::= { versionOne 2 }
brainpoolP192r1 ID ::= { versionOne 3 }
brainpoolP192t1 ID ::= { versionOne 4 }
brainpoolP224r1 ID ::= { versionOne 5 }
brainpoolP224t1 ID ::= { versionOne 6 }
brainpoolP256r1 ID ::= { versionOne 7 }
brainpoolP256t1 ID ::= { versionOne 8 }
brainpoolP320r1 ID ::= { versionOne 9 }
brainpoolP320t1 ID ::= { versionOne 10 }
brainpoolP384r1 ID ::= { versionOne 11 }
brainpoolP384t1 ID ::= { versionOne 12 }
brainpoolP512r1 ID ::= { versionOne 13 }
brainpoolP512t1 ID ::= { versionOne 14 }

X509Curves OBJECT IDENTIFIER ::= { secp192r1 | sect163k1 | sect163r2 | secp224r1 |
sect233k1 |
                                   sect233r1 | secp256r1 | sect283k1 | sect283r1 |
secp384r1 |
                                   sect409k1 | sect409r1 | secp521r1 | sect571k1 |
sect571r1 }

-- Object identifiers for Integrity Check Value (ICV) algorithms

id-hmacWithSHA224       ID ::= { digestAlgorithm 8 }  -- IETF RFC 4231
id-hmacWithSHA256       ID ::= { digestAlgorithm 9 }  -- IETF RFC 4231
id-hmacWithSHA384       ID ::= { digestAlgorithm 10 } -- IETF RFC 4231
id-hmacWithSHA512       ID ::= { digestAlgorithm 11 } -- IETF RFC 4231
```

```
id-gmac                 ID ::= { iso9797 part3(3) gmac(4) } -- ISO/IEC 9797-3

-- =============== ALGORITHMS ========================================

-- Hashing alogorithms

mD5Algorithm ALGORITHM ::= {
  PARMS           NULL
  IDENTIFIED BY {iso(1) member-body(2) us(840) rsadsi(113549) digestAlgorithm(2) md5(5)}}

-- Note that the MD5 algorithm is not considered secure

sha1Algorithm ALGORITHM ::= {
  PARMS           NULL
  IDENTIFIED BY id-sha1 }

-- Note that the SHA1 algorithm may not be considered secure

sha224 ALGORITHM ::= { -- IETF RFC 5754
  IDENTIFIED BY id-sha224 }

sha256 ALGORITHM ::= { -- IETF RFC 5754
  IDENTIFIED BY id-sha256 }

sha384 ALGORITHM ::= { -- IETF RFC 5754
 IDENTIFIED BY id-sha384 }

sha512 ALGORITHM ::= { -- IETF RFC 5754
  IDENTIFIED BY id-sha512 }

HashAlgorithms ALGORITHM ::= {sha1Algorithm |
                              sha224 |
                              sha256 |
                              sha384 |
                              sha512 }

-- Symmetric encryption algorithms

aes128-CBC ALGORITHM ::= {  -- CSOR
  PARMS          AES-InitializationVector
  IDENTIFIED BY id-aes128-CBC }

aes192-CBC ALGORITHM ::= { -- CSOR
  PARMS          AES-InitializationVector
  IDENTIFIED BY id-aes192-CBC }

aes256-CBC ALGORITHM ::= { -- CSOR
  PARMS          AES-InitializationVector
  IDENTIFIED BY id-aes256-CBC }

AES-InitializationVector ::= OCTET STRING (SIZE (16))

-- Public key algorithms

rsaEncryptionAlgorithm ALGORITHM ::= { -- IETF RFC 4055
  PARMS          NULL
  IDENTIFIED BY rsaEncryption }

keyExchangeAlgorithm ALGORITHM ::= { -- IETF RFC 3279
  PARMS          KEA-Parms-Id
  IDENTIFIED BY id-keyExchangeAlgorithm }

KEA-Parms-Id ::= OCTET STRING (SIZE (10))

dsa ALGORITHM ::= { -- IETF RFC 5480
  PARMS          DSS-Parms
  IDENTIFIED BY id-dsa }

DSS-Parms ::= SEQUENCE {
  p   INTEGER,
  q   INTEGER,
  g   INTEGER,
  ... }

ecPublicKey ALGORITHM ::= { -- IETF RFC 5480
  PARMS          X509Curves
  IDENTIFIED BY id-ecPublicKey }

ecDH ALGORITHM ::= { -- IETF RFC 5480
  PARMS          X509Curves
  IDENTIFIED BY id-ecDH }

ecMQV ALGORITHM ::= { -- IETF RFC 5480
  PARMS          X509Curves
  IDENTIFIED BY id-ecMQV }

dh-public-numberAlgorithm ALGORITHM ::= {
  PARMS          DomainParameters
  IDENTIFIED BY dh-public-number }

DomainParameters ::= SEQUENCE {
  p                INTEGER, -- odd prime, p=jq+1
  g                INTEGER, -- generator, g
  q                INTEGER, -- factor of p-1
  j                INTEGER  OPTIONAL, -- subgroup factor
  validationParms ValidationParms OPTIONAL,
  ... }

ValidationParms ::= SEQUENCE {
  seed          BIT STRING,
  pgenCounter   INTEGER,
  ... }

-- SIGNATURE ALGORITHMS
-- RSASSA-PKCS1-v1_5 signature algorithms

sha1WithRSAEncryptionAlgorithm ALGORITHM ::= { -- IETF 7427
  PARMS          NULL
  IDENTIFIED BY sha1WithRSAEncryption }

sha224WithRSAEncryptionAlgorithm ALGORITHM ::= { -- IETF RFC 5754
  PARMS          NULL
  IDENTIFIED BY sha224WithRSAEncryption }

sha256WithRSAEncryptionAlgorithm ALGORITHM ::= { -- IETF RFC 7427
  PARMS          NULL
  IDENTIFIED BY sha256WithRSAEncryption }

sha384WithRSAEncryptionAlgorithm ALGORITHM ::= { -- IETF RFC 7427
  PARMS          NULL
  IDENTIFIED BY sha384WithRSAEncryption }

sha512WithRSAEncryptionAlgorithm ALGORITHM ::= { -- IETF RFC 7427
  PARMS          NULL
  IDENTIFIED BY sha512WithRSAEncryption }

-- RSASA-PSS algorithms

rSASSA-PSS ALGORITHM ::= {
  PARMS                  SEQUENCE {
    hashAlgorithm     [0]  AlgorithmIdentifier {{HashAlgorithms}},
 -- maskGenAlgorithm [1]  AlgorithmIdentifier {{MaskGenAlgorithms}},
    saltLength        [2]  INTEGER DEFAULT 20,
    trailerField      [3]  INTEGER DEFAULT 1 }
  IDENTIFIED BY          id-RSASSA-PSS }

--

-- DSA signature algorithms

dsa-with-sha224 ALGORITHM ::= { -- IETF RFC 5754
  IDENTIFIED BY id-dsa-with-sha224 }
```

```
dsa-with-sha256 ALGORITHM ::= { -- IETF RFC 5754
  IDENTIFIED BY id-dsa-with-sha256 }

-- ECDSA signature algorithms

ecdsa-with-SHA224-Algorithm ALGORITHM ::= { -- IETF RFC
  IDENTIFIED BY ecdsa-with-SHA224 }

ecdsa-with-SHA256-Algorithm ALGORITHM ::= { -- IETF RFC 5758
  IDENTIFIED BY ecdsa-with-SHA256 }

ecdsa-with-SHA384-Algorithm ALGORITHM ::= { -- IETF RFC 5758
  IDENTIFIED BY ecdsa-with-SHA384 }

ecdsa-with-SHA512-Algorithm ALGORITHM ::= { -- IETF RFC 5758
  IDENTIFIED BY ecdsa-with-SHA512 }


-- HMAC algorithms

hmacWithSHA224 ALGORITHM ::= {  -- IETF RFC 4231
  PARMS         NULL
  IDENTIFIED BY id-hmacWithSHA224 }

hmacWithSHA256 ALGORITHM ::= {  -- IETF RFC 4231
  PARMS         NULL
  IDENTIFIED BY id-hmacWithSHA256 }

hmacWithSHA384 ALGORITHM ::= {  -- IETF RFC 4231
  PARMS         NULL
  IDENTIFIED BY id-hmacWithSHA384 }

hmacWithSHA512 ALGORITHM ::= {  -- IETF RFC 4231
  PARMS         NULL
  IDENTIFIED BY id-hmacWithSHA512 }

END -- AlgorithmObjectIdentifiers
```

Annex C

Certificate extension attribute types

(This annex forms an integral part of this Recommendation | International Standard.)

C.1 Certificate extension attribute concept

This annex includes attribute types representing certificate extension defined by this Specification and IETF RFC 5280. Such attributes will allow detailed certificate extension request to be included in a certification request as defined in IETF RFC 2986 and it allows extension information to be stored in a directory.

```
ExtensionAttribute ::= SEQUENCE {
  type              ATTRIBUTE.&id,
  value             SET SIZE (0..1) OF SEQUENCE {
    mandatory  [0]  BOOLEAN DEFAULT FALSE,
    critical   [1]  BOOLEAN DEFAULT FALSE,
    ext        [2]  EXTENSION.&ExtnType,
    ... },
  ... }
```

The **ExtensionAttribute** data type specifies the general structure of an attribute representing a certificate extension. The syntax of an attribute is a sequence with the following components:

– The **mandatory** component is a Boolean that indicates whether this extension shall be present. This component may not be relevant in certain contexts and shall then be absent.

– The **critical** component indicates whether the particular certificate extension is labelled critical or not.

– The **ext** component shall hold the actual extension syntax specification.

An extension attribute is always single-valued and it has no associated context specification.

```
extensionSyntax {EXTENSION:extension-attribute} SYNTAX-NAME ::= {
  LDAP-DESC          extension-attribute.&ldap-description
  DIRECTORY SYNTAX   SEQUENCE {
    mandatory     [0]  BOOLEAN DEFAULT FALSE,
    critical      [1]  BOOLEAN DEFAULT FALSE,
    ext           [2]  extension-attribute.&ExtnType,
    ... }
  ID                 extension-attribute.&id }
```

The **extensionSyntax** parameterized information object is to be used as a model for binding an LDAP syntax to an object identifier:

– The **LDAP-DESC** specifies the LDAP syntax for the attribute type.

– The **DIRECTORY SYNTAX** specifies the actual syntax for the attribute.

– The ID specifies the object identifier used to identify the LDAP syntax.

C.2 Formal specification for certificate extension attribute types

The formal specification is provided in the form of the ASN.1 module **ExtensionAttributes**.

```
ExtensionAttributes {joint-iso-itu-t ds(5) module(1) extensionAttributes(41) 8}
DEFINITIONS ::=
BEGIN

-- EXPORTS All

IMPORTS

  -- from Rec. ITU-T X.501 | ISO/IEC 9594-2

  attributeCertificateDefinitions, authenticationFramework, certificateExtensions,
  extensionAttributes, id-ce, informationFramework
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 8}

  ATTRIBUTE, SYNTAX-NAME
    FROM InformationFramework informationFramework
  -- from Rec. ITU-T X.509 | ISO/IEC 9594-8

  EXTENSION
     FROM AuthenticationFramework authenticationFramework

  aAissuingDistributionPoint, authorityKeyIdentifier, authorizationValidation,
  baseUpdateTime, basicConstraints, certificateIssuer, certificatePolicies,
  cRLDistributionPoints, cRLNumber, cRLStreamIdentifier, deltaCRLIndicator, deltaInfo,
  expiredCertsOnCRL, extKeyUsage, freshestCRL, holdInstructionCode, invalidityDate,
  issuerAltName, issuingDistributionPoint, keyUsage, nameConstraints, orderedList,
  policyConstraints, policyMappings, privateKeyUsagePeriod, reasonCode, revokedGroups,
  statusReferrals, subjectAltName, subjectDirectoryAttributes, subjectKeyIdentifier,
  toBeRevoked
     FROM CertificateExtensions certificateExtensions

  acceptableCertPolicies, acceptablePrivilegePolicies, allowedAttributeAssignments,
  attributeDescriptor, attributeMappings, authorityAttributeIdentifier,
  basicAttConstraints, delegatedNameConstraints, groupAC, holderNameConstraints,
  issuedOnBehalfOf, noAssertion, noRevAvail, roleSpecCertIdentifier, singleUse,
  sOAIdentifier, targetingInformation, timeSpecification, userNotice
     FROM AttributeCertificateDefinitions attributeCertificateDefinitions ;

ExtensionAttribute ::= SEQUENCE {
  type              ATTRIBUTE.&id,
  value             SET SIZE (0..1) OF SEQUENCE {
    mandatory  [0]   BOOLEAN DEFAULT FALSE,
    critical   [1]   BOOLEAN DEFAULT FALSE,
    ext        [2]   EXTENSION.&ExtnType,
    ... },
  ... }

extensionSyntax {EXTENSION:extension-attribute} SYNTAX-NAME ::= {
  LDAP-DESC          extension-attribute.&ldap-description
  DIRECTORY SYNTAX   SEQUENCE {
    mandatory    [0]   BOOLEAN DEFAULT FALSE,
    critical     [1]   BOOLEAN DEFAULT FALSE,
    ext          [2]   extension-attribute.&ExtnType,
    ... }
  ID                 extension-attribute.&id }

-- The list of extension attribute types

a-authorityKeyIdentifier ATTRIBUTE ::= {
  WITH SYNTAX        authorityKeyIdentifier.&ExtnType
  LDAP-SYNTAX        id-asx-authorityKeyIdentifier
  LDAP-NAME          {"Authority Key Identifier"}
  ID                 id-ce-a-authorityKeyIdentifier }

a-keyUsage ATTRIBUTE ::= {
  WITH SYNTAX        keyUsage.&ExtnType
  LDAP-SYNTAX        id-asx-keyUsage
  LDAP-NAME          {"Key Usage"}
  ID                 id-ce-a-keyUsage }

a-extKeyUsage ATTRIBUTE ::= {
  WITH SYNTAX        extKeyUsage.&ExtnType
  LDAP-SYNTAX        id-asx-extKeyUsage
  LDAP-NAME          {"Extended Key Usage"}
  ID                 id-ce-a-extKeyUsage }

a-privateKeyUsagePeriod ATTRIBUTE ::= {
  WITH SYNTAX        privateKeyUsagePeriod.&ExtnType
  LDAP-SYNTAX        id-asx-privateKeyUsagePeriod
  LDAP-NAME          {"Private Key Usage Period"}
  ID                 id-ce-a-privateKeyUsagePeriod }

a-certificatePolicies ATTRIBUTE ::= {
  WITH SYNTAX        certificatePolicies.&ExtnType
  LDAP-SYNTAX        id-asx-certificatePolicies
  LDAP-NAME          {"Certificate Policies"}
  ID                 id-ce-a-certificatePolicies }

a-policyMappings ATTRIBUTE ::= {
  WITH SYNTAX        policyMappings.&ExtnType
  LDAP-SYNTAX        id-asx-policyMappings
  LDAP-NAME          {"Policy Mappings"}
  ID                 id-ce-a-policyMappings }

a-authorizationValidation ATTRIBUTE ::= {
  WITH SYNTAX        authorizationValidation.&ExtnType
  LDAP-SYNTAX        id-asx-authorizationValidation
  LDAP-NAME          {"Authorization Validation"}
  ID                 id-ce-a-authorizationValidation }

a-subjectAltName ATTRIBUTE ::= {
  WITH SYNTAX        subjectAltName.&ExtnType
  LDAP-SYNTAX        id-asx-subjectAltName
  LDAP-NAME          {"Subject Alternative Name"}
  ID                 id-ce-a-subjectAltName }

a-issuerAltName ATTRIBUTE ::= {
  WITH SYNTAX        issuerAltName.&ExtnType
  LDAP-SYNTAX        id-asx-issuerAltName
  LDAP-NAME          {"Issuer Alternative Name"}
  ID                 id-ce-a-issuerAltName }

a-subjectDirectoryAttributes ATTRIBUTE ::= {
  WITH SYNTAX        subjectDirectoryAttributes.&ExtnType
  LDAP-SYNTAX        id-asx-subjectDirectoryAttributes
  LDAP-NAME          {"Subject Directory Attributes"}
  ID                 id-ce-a-subjectDirectoryAttributes }

a-basicConstraints ATTRIBUTE ::= {
  WITH SYNTAX        basicConstraints.&ExtnType
  LDAP-SYNTAX        id-asx-basicConstraints
  LDAP-NAME          {"Basic Constraints"}
  ID                 id-ce-a-basicConstraints }

a-nameConstraints ATTRIBUTE ::= {
  WITH SYNTAX        policyConstraints.&ExtnType
  LDAP-SYNTAX        id-asx-nameConstraints
  LDAP-NAME          {"Name Constraints"}
  ID                 id-ce-a-nameConstraints }

a-policyConstraints ATTRIBUTE ::= {
  WITH SYNTAX        policyConstraints.&ExtnType
  LDAP-SYNTAX        id-asx-policyConstraints
  LDAP-NAME          {"Policy Constraints"}
  ID                 id-ce-a-policyConstraints }

a-cRLNumber ATTRIBUTE ::= {
  WITH SYNTAX        cRLNumber.&ExtnType
  LDAP-SYNTAX        id-asx-cRLNumber
  LDAP-NAME          {"CRL Number"}
  ID                 id-ce-a-cRLNumber}

a-statusReferrals ATTRIBUTE ::= {
  WITH SYNTAX        statusReferrals.&ExtnType
  LDAP-SYNTAX        id-asx-statusReferrals
  LDAP-NAME          {"Status Referrals"}
  ID                 id-ce-a-statusReferrals}

a-cRLStreamIdentifier ATTRIBUTE ::= {
  WITH SYNTAX        cRLStreamIdentifier.&ExtnType
  LDAP-SYNTAX        id-asx-cRLStreamIdentifier
  LDAP-NAME          {"CRL stream identifier"}
  ID                 id-ce-a-cRLStreamIdentifier}

a-orderedList ATTRIBUTE ::= {
  WITH SYNTAX        orderedList.&ExtnType
  LDAP-SYNTAX        id-asx-orderedList
  LDAP-NAME          {"Ordered list"}
  ID                 id-ce-a-orderedList}

a-deltaInfo ATTRIBUTE ::= {
  WITH SYNTAX        deltaInfo.&ExtnType
  LDAP-SYNTAX        id-asx-deltaInfo
  LDAP-NAME          {"Delta information"}
  ID                 id-ce-a-deltaInfo}

a-toBeRevoked ATTRIBUTE ::= {
  WITH SYNTAX        toBeRevoked.&ExtnType
  LDAP-SYNTAX        id-asx-toBeRevoked
  LDAP-NAME          {"To be revoked"}
  ID                 id-ce-a-toBeRevoked}

a-revokedGroups ATTRIBUTE ::= {
  WITH SYNTAX        revokedGroups.&ExtnType
  LDAP-SYNTAX        id-asx-revokedGroups
  LDAP-NAME          {"Revoked group of certificates"}
  ID                 id-ce-a-revokedGroups}

a-expiredCertsOnCRL ATTRIBUTE ::= {
  WITH SYNTAX        expiredCertsOnCRL.&ExtnType
  LDAP-SYNTAX        id-asx-expiredCertsOnCRL
  LDAP-NAME          {"Expired certificates on CRL"}
  ID                 id-ce-a-expiredCertsOnCRL}

a-reasonCode ATTRIBUTE ::= {
  WITH SYNTAX        reasonCode.&ExtnType
  LDAP-SYNTAX        id-asx-reasonCode
  LDAP-NAME          {"Reason code"}
  ID                 id-ce-a-reasonCode}

a-holdInstructionCode ATTRIBUTE ::= {
  WITH SYNTAX        holdInstructionCode.&ExtnType
  LDAP-SYNTAX        id-asx-holdInstructionCode
  LDAP-NAME          {"Hold instruction code"}
  ID                 id-ce-a-holdInstructionCode}

a-invalidityDate ATTRIBUTE ::= {
  WITH SYNTAX        invalidityDate.&ExtnType
  LDAP-SYNTAX        id-asx-invalidityDate
  LDAP-NAME          {"Invalidity date"}
  ID                 id-ce-a-invalidityDate}

a-cRLDistributionPoints ATTRIBUTE ::= {
  WITH SYNTAX        cRLDistributionPoints.&ExtnType
  LDAP-SYNTAX        id-asx-cRLDistributionPoints
  LDAP-NAME          {"CRL distribution points"}
  ID                 id-ce-a-cRLDistributionPoints}

a-issuingDistributionPoint ATTRIBUTE ::= {
  WITH SYNTAX        issuingDistributionPoint.&ExtnType
  LDAP-SYNTAX        id-asx-issuingDistributionPoint
  LDAP-NAME          {"Issuing distribution point"}
  ID                 id-ce-a-issuingDistributionPoint}

a-certificateIssuer ATTRIBUTE ::= {
  WITH SYNTAX        certificateIssuer.&ExtnType
  LDAP-SYNTAX        id-asx-certificateIssuer
  LDAP-NAME          {"Certificate issuer"}
  ID                 id-ce-a-certificateIssuer}

a-deltaCRLIndicator ATTRIBUTE ::= {
  WITH SYNTAX        deltaCRLIndicator.&ExtnType
  LDAP-SYNTAX        id-asx-deltaCRLIndicator
  LDAP-NAME          {"Delta CRL indicator"}
  ID                 id-ce-a-deltaCRLIndicator}

a-baseUpdateTime ATTRIBUTE ::= {
  WITH SYNTAX        baseUpdateTime.&ExtnType
  LDAP-SYNTAX        id-asx-baseUpdateTime
  LDAP-NAME          {"Base update time"}
  ID                 id-ce-a-baseUpdateTime}

a-freshestCRL ATTRIBUTE ::= {
  WITH SYNTAX        freshestCRL.&ExtnType
  LDAP-SYNTAX        id-asx-freshestCRL
  LDAP-NAME          {"Freshest CRL"}
  ID                 id-ce-a-freshestCRL}

a-timeSpecification ATTRIBUTE ::= {
  WITH SYNTAX        timeSpecification.&ExtnType
  LDAP-SYNTAX        id-asx-timeSpecification
  LDAP-NAME          {"Time specification"}
  ID                 id-ce-a-timeSpecification}

a-targetingInformation ATTRIBUTE ::= {
  WITH SYNTAX        targetingInformation.&ExtnType
  LDAP-SYNTAX        id-asx-targetingInformation
  LDAP-NAME          {"Targeting information"}
  ID                 id-ce-a-targetingInformation}

a-userNotice ATTRIBUTE ::= {
  WITH SYNTAX        userNotice.&ExtnType
  LDAP-SYNTAX        id-asx-userNotice
  LDAP-NAME          {"User notice"}
  ID                 id-ce-a-userNotice}

a-acceptablePrivilegePolicies ATTRIBUTE ::= {
  WITH SYNTAX        acceptablePrivilegePolicies.&ExtnType
  LDAP-SYNTAX        id-asx-acceptablePrivilegePolicies
  LDAP-NAME          {"Acceptable Privilege Policies"}
  ID                 id-ce-a-acceptablePrivilegePolicies}

a-singleUse ATTRIBUTE ::= {
  WITH SYNTAX        singleUse.&ExtnType
  LDAP-SYNTAX        id-asx-singleUse
  LDAP-NAME          {"Single use"}
  ID                 id-ce-a-singleUse}

a-groupAC ATTRIBUTE ::= {
  WITH SYNTAX        groupAC.&ExtnType
  LDAP-SYNTAX        id-asx-groupAC
  LDAP-NAME          {"Group attribute certificate"}
  ID                 id-ce-a-groupAC}

a-noRevAvail ATTRIBUTE ::= {
  WITH SYNTAX        noRevAvail.&ExtnType
  LDAP-SYNTAX        id-asx-noRevAvail
  LDAP-NAME          {"No revocation information available"}
  ID                 id-ce-a-noRevAvail}

a-sOAIdentifier ATTRIBUTE ::= {
  WITH SYNTAX        sOAIdentifier.&ExtnType
  LDAP-SYNTAX        id-asx-sOAIdentifier
  LDAP-NAME          {"SOA identifier"}
  ID                 id-ce-a-sOAIdentifier}

a-attributeDescriptor ATTRIBUTE ::= {
  WITH SYNTAX        attributeDescriptor.&ExtnType
  LDAP-SYNTAX        id-asx-attributeDescriptor
  LDAP-NAME          {"Attribute descriptor"}
  ID                 id-ce-a-attributeDescriptor}

a-roleSpecCertIdentifier ATTRIBUTE ::= {
  WITH SYNTAX        roleSpecCertIdentifier.&ExtnType
  LDAP-SYNTAX        id-asx-roleSpecCertIdentifier
  LDAP-NAME          {"Role specification certificate identifier"}
  ID                 id-ce-a-roleSpecCertIdentifier}

a-basicAttConstraints ATTRIBUTE ::= {
  WITH SYNTAX       basicAttConstraints.&ExtnType
  LDAP-SYNTAX       id-asx-basicAttConstraints
  LDAP-NAME         {"Basic attribute constraints"}
  ID                id-ce-a-basicAttConstraints}

a-delegatedNameConstraints ATTRIBUTE ::= {
  WITH SYNTAX       delegatedNameConstraints.&ExtnType
  LDAP-SYNTAX       id-asx-delegatedNameConstraints
  LDAP-NAME         {"Delegated name constraints"}
  ID                id-ce-a-delegatedNameConstraints}

a-acceptableCertPolicies ATTRIBUTE ::= {
  WITH SYNTAX       acceptableCertPolicies.&ExtnType
  LDAP-SYNTAX       id-asx-acceptableCertPolicies
  LDAP-NAME         {"Acceptable certificate policiesGroup attribute certificate"}
  ID                id-ce-a-acceptableCertPolicies}

a-authorityAttributeIdentifier ATTRIBUTE ::= {
  WITH SYNTAX       authorityAttributeIdentifier.&ExtnType
  LDAP-SYNTAX       id-asx-authorityAttributeIdentifier
  LDAP-NAME         {"Authority attribute identifier"}
  ID                id-ce-a-authorityAttributeIdentifier}

a-indirectIssuer ATTRIBUTE ::= {
  WITH SYNTAX       indirectIssuer.&ExtnType
  LDAP-SYNTAX       id-asx-indirectIssuer
  LDAP-NAME         {"Indirect issuer"}
  ID                id-ce-a-indirectIssuer}

a-issuedOnBehalfOf ATTRIBUTE ::= {
  WITH SYNTAX       issuedOnBehalfOf.&ExtnType
  LDAP-SYNTAX       id-asx-issuedOnBehalfOf
  LDAP-NAME         {"Issued on behalf of"}
  ID                id-ce-a-issuedOnBehalfOf}

a-noAssertion ATTRIBUTE ::= {
  WITH SYNTAX       noAssertion.&ExtnType
  LDAP-SYNTAX       id-asx-noAssertion
  LDAP-NAME         {"No assertion"}
  ID                id-ce-a-noAssertion}

a-allowedAttributeAssignments ATTRIBUTE ::= {
  WITH SYNTAX       allowedAttributeAssignments.&ExtnType
  LDAP-SYNTAX       id-asx-allowedAttributeAssignments
  LDAP-NAME         {"Allowed attribute assignments"}
  ID                id-ce-a-allowedAttributeAssignments}

a-attributeMappings ATTRIBUTE ::= {
  WITH SYNTAX       attributeMappings.&ExtnType
  LDAP-SYNTAX       id-asx-attributeMappings
  LDAP-NAME         {"Attribute mappings"}
  ID                id-ce-a-attributeMappings}

a-holderNameConstraints ATTRIBUTE ::= {
  WITH SYNTAX       holderNameConstraints.&ExtnType
  LDAP-SYNTAX       id-asx-holderNameConstraints
  LDAP-NAME         {"Holder name constraints"}
  ID                id-ce-a-holderNameConstraints}

a-aAissuingDistributionPoint ATTRIBUTE ::= {
  WITH SYNTAX       aAissuingDistributionPoint.&ExtnType
  LDAP-SYNTAX       id-asx-aAissuingDistributionPoint
  LDAP-NAME         {"AA issuing distribution point"}
  ID                id-ce-a-aAissuingDistributionPoint}

-- Object identifier for attribute types

id-ce-a-subjectDirectoryAttributes         OBJECT IDENTIFIER ::= {id-ce 9 1}
id-ce-a-subjectKeyIdentifier               OBJECT IDENTIFIER ::= {id-ce 14 1}
id-ce-a-keyUsage                           OBJECT IDENTIFIER ::= {id-ce 15 1}
id-ce-a-privateKeyUsagePeriod              OBJECT IDENTIFIER ::= {id-ce 16 1}
```

```
id-ce-a-subjectAltName                    OBJECT IDENTIFIER ::= {id-ce 17 1}
id-ce-a-issuerAltName                     OBJECT IDENTIFIER ::= {id-ce 18 1}
id-ce-a-basicConstraints                  OBJECT IDENTIFIER ::= {id-ce 19 1}
id-ce-a-cRLNumber                         OBJECT IDENTIFIER ::= {id-ce 20 1}
id-ce-a-reasonCode                        OBJECT IDENTIFIER ::= {id-ce 21 1}
id-ce-a-holdInstructionCode               OBJECT IDENTIFIER ::= {id-ce 23 1}
id-ce-a-invalidityDate                    OBJECT IDENTIFIER ::= {id-ce 24 1}
id-ce-a-deltaCRLIndicator                 OBJECT IDENTIFIER ::= {id-ce 27 1}
id-ce-a-issuingDistributionPoint          OBJECT IDENTIFIER ::= {id-ce 28 1}
id-ce-a-certificateIssuer                 OBJECT IDENTIFIER ::= {id-ce 29 1}
id-ce-a-nameConstraints                   OBJECT IDENTIFIER ::= {id-ce 30 1}
id-ce-a-cRLDistributionPoints             OBJECT IDENTIFIER ::= {id-ce 31 1}
id-ce-a-certificatePolicies               OBJECT IDENTIFIER ::= {id-ce 32 1}
id-ce-a-policyMappings                    OBJECT IDENTIFIER ::= {id-ce 33 1}
id-ce-a-authorityKeyIdentifier            OBJECT IDENTIFIER ::= {id-ce 35 1}
id-ce-a-policyConstraints                 OBJECT IDENTIFIER ::= {id-ce 36 1}
id-ce-a-extKeyUsage                       OBJECT IDENTIFIER ::= {id-ce 37 1}
id-ce-a-authorityAttributeIdentifier      OBJECT IDENTIFIER ::= {id-ce 38 1}
id-ce-a-roleSpecCertIdentifier            OBJECT IDENTIFIER ::= {id-ce 39 1}
id-ce-a-cRLStreamIdentifier               OBJECT IDENTIFIER ::= {id-ce 40 1}
id-ce-a-basicAttConstraints               OBJECT IDENTIFIER ::= {id-ce 41 1}
id-ce-a-delegatedNameConstraints          OBJECT IDENTIFIER ::= {id-ce 42 1}
id-ce-a-timeSpecification                 OBJECT IDENTIFIER ::= {id-ce 43 1}
id-ce-a-cRLScope                          OBJECT IDENTIFIER ::= {id-ce 44 1}
id-ce-a-statusReferrals                   OBJECT IDENTIFIER ::= {id-ce 45 1}
id-ce-a-freshestCRL                       OBJECT IDENTIFIER ::= {id-ce 46 1}
id-ce-a-orderedList                       OBJECT IDENTIFIER ::= {id-ce 47 1}
id-ce-a-attributeDescriptor               OBJECT IDENTIFIER ::= {id-ce 48 1}
id-ce-a-userNotice                        OBJECT IDENTIFIER ::= {id-ce 49 1}
id-ce-a-sOAIdentifier                     OBJECT IDENTIFIER ::= {id-ce 50 1}
id-ce-a-baseUpdateTime                    OBJECT IDENTIFIER ::= {id-ce 51 1}
id-ce-a-acceptableCertPolicies            OBJECT IDENTIFIER ::= {id-ce 52 1}
id-ce-a-deltaInfo                         OBJECT IDENTIFIER ::= {id-ce 53 1}
id-ce-a-inhibitAnyPolicy                  OBJECT IDENTIFIER ::= {id-ce 54 1}
id-ce-a-targetingInformation              OBJECT IDENTIFIER ::= {id-ce 55 1}
id-ce-a-noRevAvail                        OBJECT IDENTIFIER ::= {id-ce 56 1}
id-ce-a-acceptablePrivilegePolicies       OBJECT IDENTIFIER ::= {id-ce 57 1}
id-ce-a-toBeRevoked                       OBJECT IDENTIFIER ::= {id-ce 58 1}
id-ce-a-revokedGroups                     OBJECT IDENTIFIER ::= {id-ce 59 1}
id-ce-a-expiredCertsOnCRL                 OBJECT IDENTIFIER ::= {id-ce 60 1}
id-ce-a-indirectIssuer                    OBJECT IDENTIFIER ::= {id-ce 61 1}
id-ce-a-noAssertion                       OBJECT IDENTIFIER ::= {id-ce 62 1}
id-ce-a-aAissuingDistributionPoint        OBJECT IDENTIFIER ::= {id-ce 63 1}
id-ce-a-issuedOnBehalfOf                  OBJECT IDENTIFIER ::= {id-ce 64 1}
id-ce-a-singleUse                         OBJECT IDENTIFIER ::= {id-ce 65 1}
id-ce-a-groupAC                           OBJECT IDENTIFIER ::= {id-ce 66 1}
id-ce-a-allowedAttributeAssignments       OBJECT IDENTIFIER ::= {id-ce 67 1}
id-ce-a-attributeMappings                 OBJECT IDENTIFIER ::= {id-ce 68 1}
id-ce-a-holderNameConstraints             OBJECT IDENTIFIER ::= {id-ce 69 1}
id-ce-a-authorizationValidation           OBJECT IDENTIFIER ::= {id-ce 70 1}

-- The list of object identifiers for LDAP syntaxes

id-asx-subjectDirectoryAttributes         OBJECT IDENTIFIER ::= {id-ce 9 2}
id-asx-subjectKeyIdentifier               OBJECT IDENTIFIER ::= {id-ce 14 2}
id-asx-keyUsage                           OBJECT IDENTIFIER ::= {id-ce 15 2}
id-asx-privateKeyUsagePeriod              OBJECT IDENTIFIER ::= {id-ce 16 2}
id-asx-subjectAltName                     OBJECT IDENTIFIER ::= {id-ce 17 2}
id-asx-issuerAltName                      OBJECT IDENTIFIER ::= {id-ce 18 2}
id-asx-basicConstraints                   OBJECT IDENTIFIER ::= {id-ce 19 2}
id-asx-cRLNumber                          OBJECT IDENTIFIER ::= {id-ce 20 2}
id-asx-reasonCode                         OBJECT IDENTIFIER ::= {id-ce 21 2}
id-asx-holdInstructionCode                OBJECT IDENTIFIER ::= {id-ce 23 2}
id-asx-invalidityDate                     OBJECT IDENTIFIER ::= {id-ce 24 2}
id-asx-deltaCRLIndicator                  OBJECT IDENTIFIER ::= {id-ce 27 2}
id-asx-issuingDistributionPoint           OBJECT IDENTIFIER ::= {id-ce 28 2}
id-asx-certificateIssuer                  OBJECT IDENTIFIER ::= {id-ce 29 2}
id-asx-nameConstraints                    OBJECT IDENTIFIER ::= {id-ce 30 2}
id-asx-cRLDistributionPoints              OBJECT IDENTIFIER ::= {id-ce 31 2}
id-asx-certificatePolicies                OBJECT IDENTIFIER ::= {id-ce 32 2}
id-asx-policyMappings                     OBJECT IDENTIFIER ::= {id-ce 33 2}
```

```
id-asx-authorityKeyIdentifier           OBJECT IDENTIFIER ::= {id-ce 35 2}
id-asx-policyConstraints                OBJECT IDENTIFIER ::= {id-ce 36 2}
id-asx-extKeyUsage                      OBJECT IDENTIFIER ::= {id-ce 37 2}
id-asx-authorityAttributeIdentifier     OBJECT IDENTIFIER ::= {id-ce 38 2}
id-asx-roleSpecCertIdentifier           OBJECT IDENTIFIER ::= {id-ce 39 2}
id-asx-cRLStreamIdentifier              OBJECT IDENTIFIER ::= {id-ce 40 2}
id-asx-basicAttConstraints              OBJECT IDENTIFIER ::= {id-ce 41 2}
id-asx-delegatedNameConstraints         OBJECT IDENTIFIER ::= {id-ce 42 2}
id-asx-timeSpecification                OBJECT IDENTIFIER ::= {id-ce 43 2}
id-asx-cRLScope                         OBJECT IDENTIFIER ::= {id-ce 44 2}
id-asx-statusReferrals                  OBJECT IDENTIFIER ::= {id-ce 45 2}
id-asx-freshestCRL                      OBJECT IDENTIFIER ::= {id-ce 46 2}
id-asx-orderedList                      OBJECT IDENTIFIER ::= {id-ce 47 2}
id-asx-attributeDescriptor              OBJECT IDENTIFIER ::= {id-ce 48 2}
id-asx-userNotice                       OBJECT IDENTIFIER ::= {id-ce 49 2}
id-asx-sOAIdentifier                    OBJECT IDENTIFIER ::= {id-ce 50 2}
id-asx-baseUpdateTime                   OBJECT IDENTIFIER ::= {id-ce 51 2}
id-asx-acceptableCertPolicies           OBJECT IDENTIFIER ::= {id-ce 52 2}
id-asx-deltaInfo                        OBJECT IDENTIFIER ::= {id-ce 53 2}
id-asx-inhibitAnyPolicy                 OBJECT IDENTIFIER ::= {id-ce 54 2}
id-asx-targetingInformation             OBJECT IDENTIFIER ::= {id-ce 55 2}
id-asx-noRevAvail                       OBJECT IDENTIFIER ::= {id-ce 56 2}
id-asx-acceptablePrivilegePolicies      OBJECT IDENTIFIER ::= {id-ce 57 2}
id-asx-toBeRevoked                      OBJECT IDENTIFIER ::= {id-ce 58 2}
id-asx-revokedGroups                    OBJECT IDENTIFIER ::= {id-ce 59 2}
id-asx-expiredCertsOnCRL                OBJECT IDENTIFIER ::= {id-ce 60 2}
id-asx-indirectIssuer                   OBJECT IDENTIFIER ::= {id-ce 61 2}
id-asx-noAssertion                      OBJECT IDENTIFIER ::= {id-ce 62 2}
id-asx-aAissuingDistributionPoint       OBJECT IDENTIFIER ::= {id-ce 63 2}
id-asx-issuedOnBehalfOf                 OBJECT IDENTIFIER ::= {id-ce 64 2}
id-asx-singleUse                        OBJECT IDENTIFIER ::= {id-ce 65 2}
id-asx-groupAC                          OBJECT IDENTIFIER ::= {id-ce 66 2}
id-asx-allowedAttributeAssignments      OBJECT IDENTIFIER ::= {id-ce 67 2}
id-asx-attributeMappings                OBJECT IDENTIFIER ::= {id-ce 68 2}
id-asx-holderNameConstraints            OBJECT IDENTIFIER ::= {id-ce 69 2}
id-asx-authorizationValidation          OBJECT IDENTIFIER ::= {id-ce 70 2}

END -- ExtensionAttributes
```

Annex D

External ASN.1 modules

(This annex does not form an integral part of this Recommendation | International Standard.)

This annex provides external ASN.1 modules referenced by this Specification and the Directory Specifications. These modules shall not be considered part of this Specification, but are only provided for easy compilation of the ASN.1 modules defined by this Specification and the Directory Specifications.

```
PkiPmiExternalDataTypes {joint-iso-itu-t ds(5) module(1) pkiPmiExternalDataTypes(40) 8}
DEFINITIONS ::=
BEGIN

-- EXPORTS All

IMPORTS

  authenticationFramework, certificateExtensions, intSecurity, selectedAttributeTypes
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 8}

  EXTENSION
    FROM AuthenticationFramework authenticationFramework

  GeneralName
    FROM CertificateExtensions certificateExtensions

  PresentationAddress
    FROM SelectedAttributeTypes selectedAttributeTypes;

/* The UserNotice data type is referenced by the userNotice extension. It is copied from
IETF RFC 5280 */

UserNotice ::= SEQUENCE {
  noticeRef     NoticeReference OPTIONAL,
  explicitText  DisplayText OPTIONAL }

NoticeReference ::= SEQUENCE {
  organization   DisplayText,
  noticeNumbers  SEQUENCE OF INTEGER }

DisplayText ::= CHOICE {
  visibleString  VisibleString(SIZE (1..200)),
  bmpString      BMPString(SIZE (1..200)),
  utf8String     UTF8String(SIZE (1..200)) }

/* IETF RFC 5280 defines some extensions not copied from this Specification. The formal
specification of these extensions are included here for easy reference, e.g., when using
ASN.1 tools. For detailed description, see IETF 5280.*/

/* Authority information access extension*/

authorityInfoAccess EXTENSION ::= {
  SYNTAX         AuthorityInfoAccessSyntax
  IDENTIFIED BY  id-pe-authorityInfoAccess }

AuthorityInfoAccessSyntax ::= SEQUENCE SIZE (1..MAX) OF AccessDescription

AccessDescription ::= SEQUENCE {
  accessMethod    OBJECT IDENTIFIER,
  accessLocation  GeneralName }

/* Subject information access extension*/

subjectInfoAccess EXTENSION ::= {
  SYNTAX         SubjectInfoAccessSyntax
  IDENTIFIED BY  id-pe-subjectInfoAccess }

SubjectInfoAccessSyntax ::= SEQUENCE SIZE (1..MAX) OF AccessDescription
```

```
/* IETF RFC 5280 (PKIX) object identifier allocation*/

id-pkix                   OBJECT IDENTIFIER ::= { intSecurity mechanisms(5) pkix(7) }

id-pe                     OBJECT IDENTIFIER ::= { id-pkix 1 }
id-ad                     OBJECT IDENTIFIER ::= { id-pkix 48 }

id-pe-authorityInfoAccess OBJECT IDENTIFIER ::= { id-pe 1 }
id-pe-subjectInfoAccess   OBJECT IDENTIFIER ::= { id-pe 11 }
id-ad-caIssuers           OBJECT IDENTIFIER ::= { id-ad 2 }
id-ad-ocsp                OBJECT IDENTIFIER ::= { id-ad 1 }


/* The following is an abstract of the MTSAbstractService module specified by
Rec. ITU-T X.411 | ISO/IEC 10021-4.*/

G3FacsimileNonBasicParameters ::= BIT STRING {
  two-dimensional(8), -- As defined in ITU-T Recommendation T.30
  fine-resolution(9),
  unlimited-length(20), -- These bit values are chosen such that when
  b4-length(21), -- encoded using ASN.1 Basic Encoding Rules
  a3-width(22), -- the resulting octets have the same values
  b4-width(23), -- as for T.30 encoding
  t6-coding(25),
  uncompressed(30), -- Trailing zero bits are not significant
  width-middle-864-of-1728(37), -- It is recommended that implementations
  width-middle-1216-of-1728(38), -- should not encode more than 32 bits unless
  resolution-type(44), -- higher numbered bits are non-zero
  resolution-400x400(45), resolution-300x300(46), resolution-8x15(47),
  edi(49), dtm(50), bft(51), mixed-mode(58), character-mode(60),
  twelve-bits(65), preferred-huffmann(66), full-colour(67), jpeg(68),
  processable-mode-26(71)}

ORAddress ::= SEQUENCE {
  built-in-standard-attributes        BuiltInStandardAttributes,
  built-in-domain-defined-attributes  BuiltInDomainDefinedAttributes OPTIONAL,
  -- see also teletex-domain-defined-attributes
  extension-attributes                ExtensionAttributes OPTIONAL }

--     The OR-address is semantically absent from the OR-name if the built-in-standard-
attribute
--     sequence is empty and the built-in-domain-defined-attributes and extension-
attributes are both omitted.
--     Built-in Standard Attributes

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

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

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

NetworkAddress ::= X121Address

-- see also extended-network-address
```

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

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

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

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

-- see also teletex-organization-name

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

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

-- see also teletex-personal-name

OrganizationalUnitNames ::=
  SEQUENCE SIZE (1..ub-organizational-units) OF OrganizationalUnitName

-- see also teletex-organizational-unit-names
OrganizationalUnitName ::=
  PrintableString(SIZE (1..ub-organizational-unit-name-length))

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

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

--     Extension Attributes

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

ExtensionAttribute ::= SEQUENCE {
  extension-attribute-type
    [0]  EXTENSION-ATTRIBUTE.&id({ExtensionAttributeTable}),
  extension-attribute-value
    [1]  EXTENSION-ATTRIBUTE.&Type
           ({ExtensionAttributeTable}{@extension-attribute-type}) }

EXTENSION-ATTRIBUTE ::= CLASS {
  &id            INTEGER(0..ub-extension-attributes) UNIQUE,
  &Type }
WITH SYNTAX {
                 &Type
  IDENTIFIED BY &id }

ExtensionAttributeTable EXTENSION-ATTRIBUTE ::=
  {common-name | teletex-common-name | universal-common-name |
   teletex-organization-name | universal-organization-name |
   teletex-personal-name | universal-personal-name |
   teletex-organizational-unit-names | universal-organizational-unit-names |
   teletex-domain-defined-attributes | universal-domain-defined-attributes |
   pds-name | physical-delivery-country-name | postal-code |
   physical-delivery-office-name | universal-physical-delivery-office-name |
   physical-delivery-office-number | universal-physical-delivery-office-number
   | extension-OR-address-components |
   universal-extension-OR-address-components | physical-delivery-personal-name
   | universal-physical-delivery-personal-name |
   physical-delivery-organization-name |
   universal-physical-delivery-organization-name |
   extension-physical-delivery-address-components |
   universal-extension-physical-delivery-address-components |
   unformatted-postal-address | universal-unformatted-postal-address |
   street-address | universal-street-address | post-office-box-address |
   universal-post-office-box-address | poste-restante-address |
   universal-poste-restante-address | unique-postal-name |
   universal-unique-postal-name | local-postal-attributes |
   universal-local-postal-attributes | extended-network-address | terminal-type }

--     Extension Standard Attributes

common-name EXTENSION-ATTRIBUTE ::= {
                 CommonName
  IDENTIFIED BY  1 }

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

teletex-common-name EXTENSION-ATTRIBUTE ::= {
                 TeletexCommonName
  IDENTIFIED BY  2 }

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

universal-common-name EXTENSION-ATTRIBUTE ::= {
                 UniversalCommonName
  IDENTIFIED BY  24 }

UniversalCommonName ::= UniversalOrBMPString{ub-common-name-length}

teletex-organization-name EXTENSION-ATTRIBUTE ::= {
                 TeletexOrganizationName
  IDENTIFIED BY  3 }

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

universal-organization-name EXTENSION-ATTRIBUTE ::= {
                 UniversalOrganizationName
  IDENTIFIED BY  25 }

UniversalOrganizationName ::= UniversalOrBMPString{ub-organization-name-length}

teletex-personal-name EXTENSION-ATTRIBUTE ::= {
                 TeletexPersonalName
  IDENTIFIED BY  4 }

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

universal-personal-name EXTENSION-ATTRIBUTE ::= {
                 UniversalPersonalName
  IDENTIFIED BY  26 }

UniversalPersonalName ::= SET {
  surname
    [0]  UniversalOrBMPString{ub-universal-surname-length},
  -- If a language is specified within surname, then that language applies to each of the
  -- following optional components unless the component specifies another language.
  given-name
    [1]  UniversalOrBMPString{ub-universal-given-name-length} OPTIONAL,
  initials
```