INTERNATIONAL STANDARD

ISO/IEC 9594-8

Seventh edition
2014-03-01

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 9594-8:2014(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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.

ISO/IEC 9594-8 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 6, *Telecommunications and information exchange between systems*, in collaboration with ITU-T. The identical text is published as Rec. ITU-T X.509 (10/2012).

This seventh edition cancels and replaces the sixth edition (ISO/IEC 9594-8:2008), which has been technically revised. It also incorporates the Technical Corrigenda ISO/IEC 9594-8:2008/Cor.1:2011, ISO/IEC 9594-8:2008/Cor.2:2012 and ISO/IEC 9594-8:2008/Cor.3:2013.

ISO/IEC 9594 consists of the following parts, under the general title *Information technology — Open Systems Interconnection — The Directory*:

— *Part 1: Overview of concepts, models and services*

— *Part 2: Models*

— *Part 3: Abstract service definition*

— *Part 4: Procedures for distributed operation*

— *Part 5: Protocol specifications*

— *Part 6: Selected attribute types*

— *Part 7: Selected object classes*

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

— *Part 9: Replication*

CONTENTS

Introduction

This Recommendation | International Standard, together with other Recommendations | International Standards, has been produced to facilitate the interconnection of information processing systems to provide directory services. A set of such systems, together with the directory information which they hold, can be viewed as an integrated whole, called the *Directory*. The information held by the Directory, collectively known as the Directory Information Base (DIB), is typically used to facilitate communication between, with or about objects such as application-entities, people, terminals and distribution lists.

The Directory plays a significant role in Open Systems Interconnection, whose aim is to allow, with a minimum of technical agreement outside of the interconnection standards themselves, the interconnection of information processing systems:

– from different manufacturers;
– under different managements;
– of different levels of complexity; and
– of different ages.

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., authentication.

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

Similarly, this Recommendation | International Standard defines a framework for attribute certificates. That framework includes the specification of data objects used to represent the certificates themselves, as well as revocation notices for issued certificates that should no longer be trusted. The attribute certificate framework defined in this Recommendation | International Standard, while it defines some critical components of a Privilege Management Infrastructure (PMI), it does not define a PMI in its entirety. However, this Recommendation | International Standard provides the foundation upon which full PMIs and their specifications would be built.

Information objects for holding PKI and PMI objects in the Directory and for comparing presented values with stored values are also defined.

This Recommendation | International Standard also defines a framework for the provision of authentication services by the Directory to its users.

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 seventh edition technically revises and enhances the sixth edition of this Recommendation | International Standard.

This seventh edition specifies versions 1, 2 and 3 of public-key certificates and versions 1 and 2 of certificate revocation lists. This edition also specifies 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. This function is specified in clause 7. It is anticipated that any enhancements to this edition can be accommodated using this function and avoid the need to create new versions.

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 an integral part of this Recommendation | International Standard, defines object identifiers assigned to authentication and encryption algorithms, in the absence of a formal register.

Annex C, which is an integral part of this Recommendation | International Standard, provides rules for generating and processing Certificate Revocation Lists.

Annex D, which is not an integral part of this Recommendation | International Standard, provides examples of delta-CRL issuance.

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

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

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

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

Annex I, 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 J, 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 K, which is not an integral part of this Recommendation | International Standard, provides a suggested technique for a Bind protected password.

Annex L, which is not an integral part of this Recommendation | International Standard, gives some examples of password hashing algorithms.

Annex M, 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 N, 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
RECOMMENDATION ITU-T

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

SECTION 1 – GENERAL

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;
- attribute certificates; and
- authentication services.

The public-key certificate framework defined in this Recommendation | International Standard includes a definition of the information objects for a public-key infrastructure (PKI), including public-key certificates and Certificate Revocation Lists (CRLs). The attribute certificate framework includes a definition of the information objects 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 standard extensions for each, 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 objects in the Directory, are included in this Recommendation | International Standard. Other elements of PKI and PMI, beyond these frameworks, such as key and certificate management protocols, operational protocols, additional certificate and CRL extensions are expected to be defined by other standards bodies (e.g., ISO TC 68, IETF, etc.).

The authentication scheme defined in this Recommendation | International Standard is generic and may be applied to a variety of applications and environments.

The Directory makes use of public-key certificates and attribute certificates, and the framework for the Directory's use of these facilities is also defined in this Recommendation | International Standard. Public-key technology, including certificates, is used by the Directory to enable strong authentication and signed operations, and for storage of signed data in the Directory. Attribute certificates can be used by the Directory to enable rule-based access control. Although the framework for these is provided in this Recommendation | International Standard, the full definition of the Directory's use of these frameworks, and the associated services provided by the Directory and its components is supplied in the complete set of ITU-T X.500 series of Recommendations | ISO/IEC 9594 (all parts).

This Recommendation | International Standard, in the Authentication services framework, also:

- specifies the form of authentication information held by the Directory;
- describes how authentication information may be obtained from the Directory;
- states the assumptions made about how authentication information is formed and placed in the Directory;
- defines three ways in which applications may use this authentication information to perform authentication and describes how other security services may be supported by authentication.

This Recommendation | International Standard describes two levels of authentication: simple authentication, using a password as a verification of claimed identity; and strong authentication, involving credentials formed using cryptographic techniques. While simple authentication offers some limited protection against unauthorized access, only strong authentication should be used as the basis for providing secure services. 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 which may be constrained by the services provided by a standard.

It is a matter for standards-defining applications which use the authentication framework to specify the protocol exchanges which need to be performed in order to achieve authentication based upon the authentication information obtained from the Directory. The protocol used by applications to obtain credentials from the Directory is the Directory Access Protocol (DAP), specified in Rec. ITU-T X.519 | ISO/IEC 9594-5.

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 (2012) | ISO/IEC 9594-1:2014, *Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services*.
- Recommendation ITU-T X.501 (2012) | ISO/IEC 9594-2:2014, *Information technology – Open Systems Interconnection – The Directory: Models*.
- Recommendation ITU-T X.511 (2012) | ISO/IEC 9594-3:2014, *Information technology – Open Systems Interconnection – The Directory: Abstract service definition*.
- Recommendation ITU-T X.518 (2012) | ISO/IEC 9594-4:2014, *Information technology – Open Systems Interconnection – The Directory: Procedures for distributed operation*.
- Recommendation ITU-T X.519 (2012) | ISO/IEC 9594-5:2014, *Information technology – Open Systems Interconnection – The Directory: Protocol specifications*.
- Recommendation ITU-T X.520 (2012) | ISO/IEC 9594-6:2014, *Information technology – Open Systems Interconnection – The Directory: Selected attribute types*.
- Recommendation ITU-T X.521 (2012) | ISO/IEC 9594-7:2014, *Information technology – Open Systems Interconnection – The Directory: Selected object classes*.
- Recommendation ITU-T X.525 (2012) | ISO/IEC 9594-9:2014, *Information technology – Open Systems Interconnection – The Directory: Replication*.
- Recommendation ITU-T X.660 (2008) | ISO/IEC 9834-1:2008, *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.680 (2008) | ISO/IEC 8824-1:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.
- Recommendation ITU-T X.681 (2008) | ISO/IEC 8824-2:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.
- Recommendation ITU-T X.682 (2008) | ISO/IEC 8824-3:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification*.
- Recommendation ITU-T X.683 (2008) | ISO/IEC 8824-4:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.
- Recommendation ITU-T X.690 (2008) | ISO/IEC 8825-1:2008, *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.691 (2008) | ISO/IEC 8825-2:2008, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*.
- 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 CCITT 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 1035 (1987), *Domain names – implementation and specification*.

– 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 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*.

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 CCITT Rec. X.800 | ISO 7498-2:

a) asymmetric (encipherment);
b) authentication exchange;
c) authentication information;
d) confidentiality;
e) credentials;
f) cryptography;
g) data origin authentication;
h) decipherment;
i) digital signature;
j) encipherment;
k) key;
l) password;
m) peer-entity authentication;
n) symmetric (encipherment).

3.2 Baseline identity management terms and definitions

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

a) trust: The firm belief in the reliability and truth of information or in the ability and disposition of an entity to act appropriately, within a specified context.

3.3 Directory model definitions

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

a) attribute;
b) Directory Information Base;
c) Directory Information Tree;
d) Directory System Agent;
e) Directory User Agent;
f) distinguished name;
g) entry;
h) object;
i) 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 attribute certificate (AC): A data structure, digitally signed by an Attribute Authority, that binds some attribute values with identification information about its holder.

3.5.2 Attribute Authority (AA): An authority which assigns privileges by issuing attribute certificates.

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

3.5.4 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 authority.

3.5.5 authentication token; (token): Information conveyed during a strong authentication exchange, which can be used to authenticate its sender.

3.5.6 authority: An entity, responsible for the issuance of certificates. Two types are defined in this Recommendation | International Standard; a certification authority which issues public-key certificates and an attribute authority which issues attribute certificates.

3.5.7 authority certificate: A certificate issued to an authority (e.g., either to a certification authority or to an attribute authority).

3.5.8 base CRL: A CRL that is used as the foundation in the generation of a dCRL.

3.5.9 CA-certificate: A public-key certificate for one CA issued by either another CA or by the same CA.

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

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

3.5.12 certificate revocation list (CRL): A signed list indicating a set of certificates that are no longer considered valid by the certificate issuer. In addition to the generic term CRL, some specific CRL types are defined for CRLs that cover particular scopes.

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

3.5.14 certificate-using system: An implementation of those functions defined in this Recommendation | International Standard that are used by a relying party.

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

3.5.16 certification authority (CA): An authority trusted by one or more users to create and assign public-key certificates. Optionally the certification authority may create the subjects' keys.

3.5.17 certification authority revocation list (CARL): A revocation list containing a list of CA-certificates issued to certification authorities that are no longer considered valid by the certificate issuer.

3.5.18 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 public key certificate to be validated. All intermediate public-key certificates, if any, are CA-certificates in which the subject of the preceding certificate is the issuer of the following certificate.

3.5.19 CRL distribution point: A directory entry or other distribution source for CRLs; a CRL distributed through a CRL distribution point may contain revocation entries for only a subset of the full set of certificates issued by one CA or may contain revocation entries for multiple CAs.

3.5.20 cross-certificate: A public-key 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.

3.5.21 cryptographic system, cryptosystem: A collection of transformations from plain text into cipher text and vice versa, the particular transformation(s) to be used being selected by keys. The transformations are normally defined by a mathematical algorithm.

3.5.22 data confidentiality: This service can be used to provide the protection of data from unauthorized disclosure. The data confidentiality service is supported by the authentication framework. It can be used to protect against data interception.

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

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

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

3.5.26 end-entity: Either a public-key certificate subject that uses its private key for purposes other than signing certificates, or an attribute certificate holder that uses its attributes to gain access to a resource.

3.5.27 end-entity attribute certificate: An attribute certificate issued to an end-entity.

3.5.28 end-entity attribute certificate revocation list (EARL): A revocation list containing a list of end-entity attribute certificates that are no longer considered valid by the issuing attribute authority.

3.5.29 end-entity certificate: An attribute or public-key certificate issued to an end-entity.

3.5.30 end-entity public-key certificate: A public-key certificate issued to an end-entity.

3.5.31 end-entity public-key certificate revocation list (EPRL): A revocation list containing a list of end-entity public-key certificates that are no longer considered valid by the issuing certification authority.

3.5.32 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).

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

3.5.34 hash function: A (mathematical) function which maps values from a large (possibly very large) domain into a smaller range. A "good" hash function is such that the results of applying the function to a (large) set of values in the domain will be evenly distributed (and apparently at random) over the range.

3.5.35 holder: An entity to whom some privilege has been delegated either directly from the Source of Authority or indirectly through another Attribute Authority.

3.5.36 indirect CRL (iCRL): A revocation list that contains at least revocation information about certificates issued by authorities other than that which issued this CRL.

3.5.37 key agreement: A method for negotiating a key value online 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).

3.5.38 password expiration: A situation where a user password has reached the end of its validity period; the account is locked and the user has to change the password before doing any other directory operation.

3.5.39 password quality attributes: Attributes that specify how a password shall be constructed. Password quality attributes include things like minimum length, mixture of characters (uppercase, lowercase, figures, punctuation, etc.), and avoidance of trivial passwords.

3.5.40 password history: A list of old passwords and the times they were inserted in the history.

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

3.5.42 one-way function: A (mathematical) function f 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 of y for which finding x is not computationally difficult.

3.5.43 policy decision point (PDP): The point where policy decisions are made (synonymous with ADF).

3.5.44 policy enforcement point (PEP): The point where the policy decisions are actually enforced (synonymous with AEF).

3.5.45 policy mapping: Recognizing that, when a 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.

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

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

3.5.48 privilege asserter: A privilege holder using their attribute certificate or public-key certificate to assert privilege.

3.5.49 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.

3.5.50 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.

3.5.51 privilege verifier: An entity verifying certificates against a privilege policy.

3.5.52 public-key: (In a public key cryptosystem) that key of a user's key pair which is publicly known.

3.5.53 public-key certificate (PKC): The public key of a user, together with some other information, rendered unforgeable by digital signature with the private key of the CA which issued it.

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

3.5.55 relying party: A user or agent that relies on the data in a certificate in making decisions.

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

3.5.57 role specification certificate: A certificate that contains the assignment of privileges to a role.

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

3.5.59 simple authentication: Authentication by means of simple password arrangements.

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

3.5.61 self-issued attribute certificate: An attribute certificate where the issuer and the subject are the same Attribute Authority. An Attribute Authority might use a self-issued AC, for example, to publish policy information.

3.5.62 self-issued certificate: A public-key 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.

3.5.63 self-signed certificate: 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 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 CAs are outside the scope of this Recommendation | International Standard.

3.5.64 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.

3.5.65 strong authentication: Authentication by means of cryptographically derived credentials.

3.5.66 trust anchor: A trust anchor is an entity that is trusted by a relying party and used for validating certificates in certification paths.

3.5.67 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 CA-certificate or as a normal CA-certificate (i.e., cross-certificate).

4 Abbreviations

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

AA	Attribute Authority
AARL	Attribute Authority Revocation List
AC	Attribute Certificate
ACRL	Attribute Certificate Revocation List
ADF	Access control Decision Function
AEF	Access control Enforcement Function
AIA	Authority Information Access
CA	Certification Authority
CARL	Certification Authority Revocation List
CRL	Certificate Revocation List
DAP	Directory Access Protocol
dCRL	Delta Certificate Revocation List
DIB	Directory Information Base
DIT	Directory Information Tree
DS	Delegation Service
DSA	Directory System Agent
DUA	Directory User Agent
EARL	End-entity Attribute certificate Revocation List
EPRL	End-entity Public-key certificate Revocation List
IAI	Issuer's ACs Identifiers
iCRL	Indirect Certificate Revocation List
OCSP	Online Certificate Status Protocol
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PKC	Public-Key Certificate
PKCS	Public-Key Cryptosystem
PKI	Public-Key Infrastructure

PMI	Privilege Management Infrastructure
RoA	Recognition of Authority
SOA	Source of Authority

5 Conventions

The term "Directory Specification" (as in "this Directory Specification") shall be taken to mean Rec. ITU-T X.509 | ISO/IEC 9594-8. The term "Directory Specifications" shall be taken to mean the ITU-T X.500-series Recommendations and all parts of ISO/IEC 9594.

This Directory Specification uses the term *first edition systems* to refer to systems conforming to the first edition of these Directory Specifications, i.e., the 1988 edition of the series of CCITT X.500 Recommendations and the ISO/IEC 9594:1990 edition.

This Directory Specification uses the term *second edition systems* to refer to systems conforming to the second edition of these Directory Specifications, i.e., the 1993 edition of the series of ITU-T X.500 Recommendations and the ISO/IEC 9594:1995 edition.

This Directory Specification uses the term *third edition systems* to refer to systems conforming to the third edition of these Directory Specifications, i.e., the 1997 edition of the series of ITU-T X.500 Recommendations and the ISO/IEC 9594:1998 edition.

This Directory Specification uses the term *fourth edition systems* to refer to systems conforming to the fourth edition of the Directory Specifications, i.e., the 2001 editions of Recs ITU-T X.500, X.501, X.511, X.518, X.519, X.520, X.521, X.525, and X.530, the 2000 edition of Rec. ITU-T X.509, and parts 1-10 of the ISO/IEC 9594:2001 edition.

This Directory Specification uses the term *fifth edition systems* to refer to systems conforming to the fifth edition of these Directory Specifications, i.e., the 2005 edition of the series of ITU-T X.500 Recommendations and the ISO/IEC 9594:2005 edition.

This Directory Specification uses the term sixth edition systems to refer to systems conforming to the sixth edition of these Directory Specifications, i.e., the 2008 edition of the series of ITU-T X.500 Recommendations and the ISO/IEC 9594:2008 edition.

This Directory Specification uses the term seventh edition systems to refer to systems conforming to the seventh edition of these Directory Specifications, i.e., the 2012 edition of the series of ITU-T X.500 Recommendations and the ISO/IEC 9594:2014 edition.

This Directory 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. The names of procedures, typically referenced when specifying the semantics of processing, are differentiated from normal text by displaying them in bold Times New Roman. Access control permissions are presented in italicized Times New Roman. When a definition is referenced for the first time in normal text it is also 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 Directory Specification defines a framework for obtaining and trusting a public key of an entity in order to encrypt information to be decrypted by that entity, or in order to verify the digital signature of that entity. 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 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 were not expired or not revoked at a given time).

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

Although PKI and PMI are separate infrastructures and may be established independently from one another, they are related. This Directory 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.

This Directory Specification makes extensive use of public-key cryptography. Annex F introduces this technology.

6.1 Digital signatures

Digital signatures are used in both PKI and PMI as the mechanism by which the authority that issues a certificate certifies the binding in the certificate. In PKI, the digital signature of the issuing CA on a public-key certificate certifies the binding between the public-key material and the subject of the public-key certificate. In PMI, the digital signature of the issuing AA certifies the binding between the attributes (privileges) and the holder of the certificate. This subclause describes digital signatures in general. Sections 2 and 3 of this Directory Specification discuss the use of digital signatures within PKI and PMI specifically.

This subclause is not intended to specify a standard for digital signatures in general, but to specify the means by which instances of the PKI and PMI specific data types are signed.

Information (info) is signed by appending to it an enciphered summary of the information. The summary is produced by means of a one-way hash function, while the enciphering is carried out using the private key of the signer (see Figure 1). Thus:

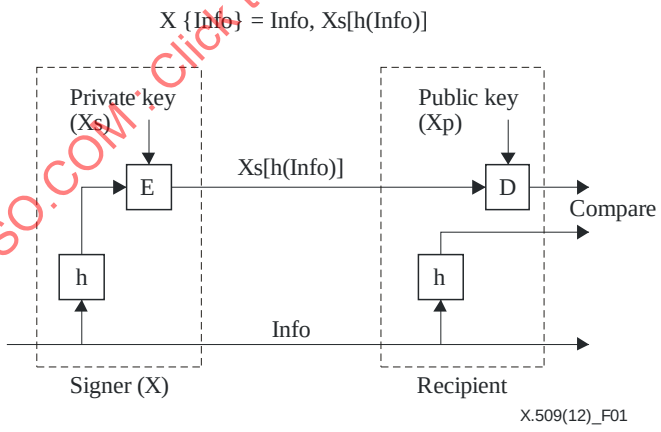


Figure 1 – Digital signatures

NOTE – The encipherment using the private key ensures that the signature cannot be forged. The one-way nature of the hash function ensures that false information, generated to have the same hash result (and thus signature) cannot be substituted.

The recipient of signed information verifies the signature by:

- applying the one-way hash function to the information;
- comparing the result with that obtained by deciphering the signature using the public key of the signer.

This Directory Specification does not mandate a single one-way hash function for use in signing. It is intended that the framework shall be applicable to any suitable hash function, and shall thus support changes to the methods used because of future advances in cryptography, mathematical techniques or computational capabilities. However, two users wishing to authenticate shall support the same hash function for authentication to be performed correctly. Thus, within the

context of a set of related applications, the choice of a single function shall serve to maximize the community of users able to authenticate and communicate securely.

6.2 Formal definitions for public-key cryptography

The encipherment of a data item may be described using the following ASN.1:

```
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 } )
```

The value of the bit string is generated by taking the octets which form the complete encoding (using the ASN.1 Basic Encoding Rules – Rec. ITU-T X.690 | ISO/IEC 8825-1) of the value of the **ToBeEnciphered** type and applying an encipherment procedure to those octets.

NOTE 1 – The encryption procedure requires agreement on the algorithm to be applied, including any parameters of the algorithm such as any necessary keys, initialization values, and padding instructions. It is the responsibility of the encryption procedures to specify the means by which synchronization of the sender and receiver of data is achieved, which may include information in the bits to be transmitted.

NOTE 2 – The encryption procedure is required to take as input a string of octets and to generate a single string of bits as its result.

NOTE 3 – Mechanisms for secure agreement on the encryption algorithm and its parameters by the sender and receiver of data are outside the scope of this Directory Specification.

The signature of a data item is formed by encrypting a shortened or "hashed" transformation of the item, and may be described by the following ASN.1:

```
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-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 -- } )

SIGNATURE{ToBeSigned} ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}},
  encrypted            ENCRYPTED-HASH{ToBeSigned},
  ... }
```

NOTE 4 – The encryption procedure requires the agreements listed in Note 1, and agreement as to whether the hashed octets are encrypted directly, or only after further encoding them as a BIT STRING using the ASN.1 Basic Encoding Rules.

In the case where a signature is appended to a data type, 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{ToBeSigned},
  ... }
```

6.3 Distinguished encoding of Basic Encoding Rules

In order to enable the validation of **SIGNED** and **SIGNATURE** types in a distributed environment, a distinguished encoding is required. A distinguished encoding of a **SIGNED** or **SIGNATURE** data value 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 value;

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

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 UTC time shall be as specified in Rec. ITU-T X.690 | ISO/IEC 8825-1;

j) the encoding of a Generalized time shall be as 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 contains unknown protocol extensions or unknown attribute syntaxes. The entity shall follow these rules:

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

– 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.

– 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.

SECTION 2 – PUBLIC-KEY CERTIFICATE FRAMEWORK

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

A relying party needs to validate a public-key certificate prior to using that public-key certificate for an application. Procedures for performing that validation are also defined here, including verifying the integrity of the public-key certificate itself, its revocation status, and its validity with respect to the intended use.

The Directory uses public-key certificates in its provision of security services including:

– strong authentication between and among directory components;

– authentication and integrity of directory operations; as well as

– integrity and authentication of stored data.

7 Public-keys and public-key certificates

7.1 Introduction

In order for a user to be able to trust a public-key for another user, for instance to authenticate the identity of that user, 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 Directory Specification. The certificate, the form of which is specified later in this clause, has the following properties:

– any user 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 certificate without this being detected (certificates are unforgeable).

Because certificates are unforgeable, they can be published by being placed in the Directory, without the need for the latter to make special efforts to protect them.

NOTE – Although the CAs are unambiguously defined by a distinguished name in the DIT, this does not imply that there is any relationship between the organization of the CAs and the DIT.

7.2 Public-key certificate

A CA issues a public-key certificate of an entity by signing (see clause 6.1) a collection of information, including its distinguished name, the user'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 can be used to represent 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

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

-- Definition of the following information object set is deferred, perhaps to
-- standardized profiles or to protocol implementation conformance statements. The
-- set is required to specify a table constraint on the parameters component of
-- AlgorithmIdentifier.

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

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

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

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

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

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

Extensions ::= SEQUENCE 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 field 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 certificates defined in this Directory 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.

The **version** field 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** field 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 identify a unique public-key certificate).

The **signature** field contains the algorithm identifier for the algorithm and hash function used by the CA in signing the certificate (e.g., md5WithRSAEncryption, sha-1WithRSAEncryption, id-dsa-with-sha1, 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 2 – This field is redundant except possibly for its participation in matching certificates (see clause 11.3.2).

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

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

The **subject** field shall identify the entity associated with the public-key found in the **subjectPublicKey** component of the **subjectPublicKeyInfo** field. If the public-key certificate is for an end-entity, then the distinguished name may be an empty sequence providing that the **subjectAltName** extension is present and is flagged as critical. Otherwise, it shall be a non-empty distinguished name (see clause 8.3.2.1).

The **subjectPublicKeyInfo** field consists of two components:

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

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

The **issuerUniqueIdentifier** field is used uniquely to identify an issuer in case of name reuse.

The **subjectUniqueIdentifier** field is used uniquely to identify a subject in case of name reuse.

NOTE 3 – The use of issuerUniqueIdentifier and the subjectUniqueIdentifier is deprecated. These fields were added because at one time there was some fear of the reuse of distinguished names.

A user may obtain one or more public-key certificates from one or more CAs. Each certificate bears the name of the CA which issued it.

7.3 Public-key certificate extensions

The **extensions** field allows for the addition of new fields to the structure without modification to the ASN.1 definition. An extension field 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** 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 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 certificate, an unrecognized critical extension would 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 marked 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.

Note that any extension that is flagged 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 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:

i) it can exclude the extension from the certificate;

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

iii) it can include the extension and flag it critical.

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

i) if the extension is unrecognized and is marked non-critical, the relying party shall ignore the extension and accept the certificate (all other things being equal);

ii) if the extension is unrecognized and marked critical, the relying party shall reject the certificate;

iii) if the extension is recognized, the relying party shall process the extension and accept or reject the certificate 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 can only be marked critical. In these cases, a relying party that understands the extension processes it; the acceptance/rejection of the certificate is dependent (at least in part) on the content of the extension. A relying party that does not understand the extension shall reject the certificate.

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

Some extensions may be marked critical or non-critical. In these cases, a relying party that understands the extension processes it: the acceptance/rejection of the 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 certificate if the extension is marked non-critical (unless factors other than this extension cause it to be rejected) and rejects the certificate if the extension is marked critical.

When a CA considers including an extension in a 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 critical. This is done with the realization that any relying party that does not process the extension will reject the certificate (probably limiting the set of applications that can verify the 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 marked non-critical. It is most likely that CAs would set optionally critical extensions as non-critical during a transition period while the verifiers' 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. Standard extensions for public-key certificates are defined in clause 8 of this Directory Specification.

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 a subject that is not an issuer of other public-key certificates.

A CA-certificate is a public-key certificate issued by a CA to a subject that is also a CA and therefore is capable of issuing public-key certificates. A CA-certificate shall include the **basicConstraints** extension with the **cA** components set to **TRUE** (see clause 8.4.2.1).

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 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

A trust anchor is an entity that is trusted for the purpose of certificate validation by a relying party. Information about a trust anchor (trust anchor information) is typically configured into the relying party in a so-called trust anchor store. A relying party may have configured information about multiple trust anchors into one or more trust anchor storages.

NOTE – Trust anchor has in the past been synonymous with the term root-CA. In a strict hierarchy, the CA at the top of the hierarchy is called the root CA and it may be the 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 trust anchor information may be provided as:

- a self-signed certificate, or
- a normal CA cross-certificate.

7.6 Entity relationship

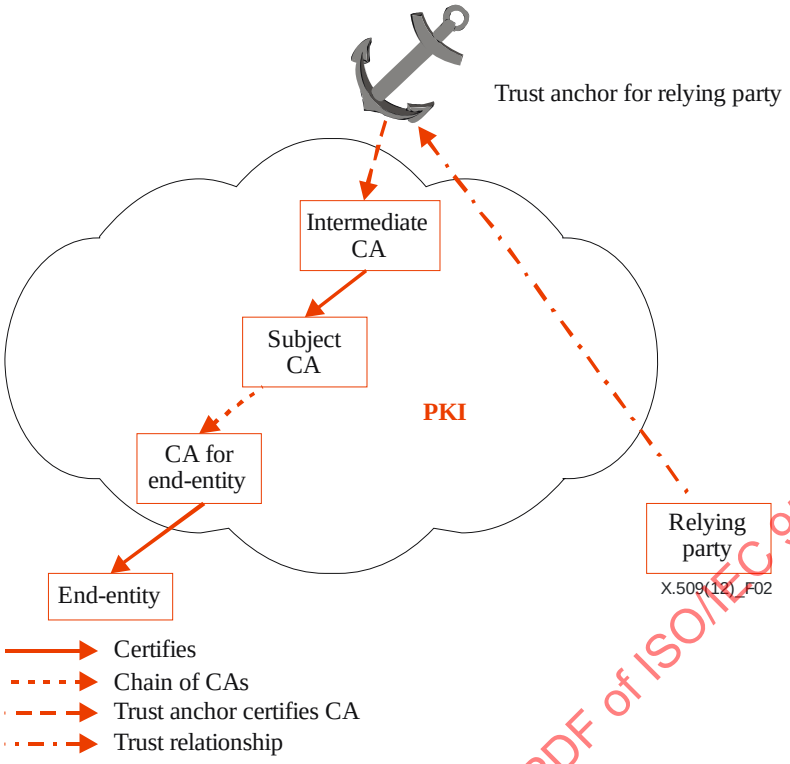


Figure 2 – Entity relationships

There may be several CAs between the trust anchor recognized by the relying party and an end-entity for which a public-key certificate is to be validated. 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.

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 certificates are established between the end-entity and the trust anchor.

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-keycertificate 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 public key certificate to be validated. All intermediate public-key certificates, if any, are CA-certificates in which the subject of the preceding certificate is the issuer of the following certificate.

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

The **`issuer`** and **`subject`** fields of each 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`** field in one certificate shall match the value of the **`issuer`** field in the subsequent certificate. In addition, the value of the **`issuer`** field in the public-key certificate issued by the trust anchor shall match the distinguished name of the trust anchor. Only the names in these fields are used when checking the validity of a certification path. Names in 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 **`issuer`** field of one certificate with the distinguished name in the **`subject`** field of another.

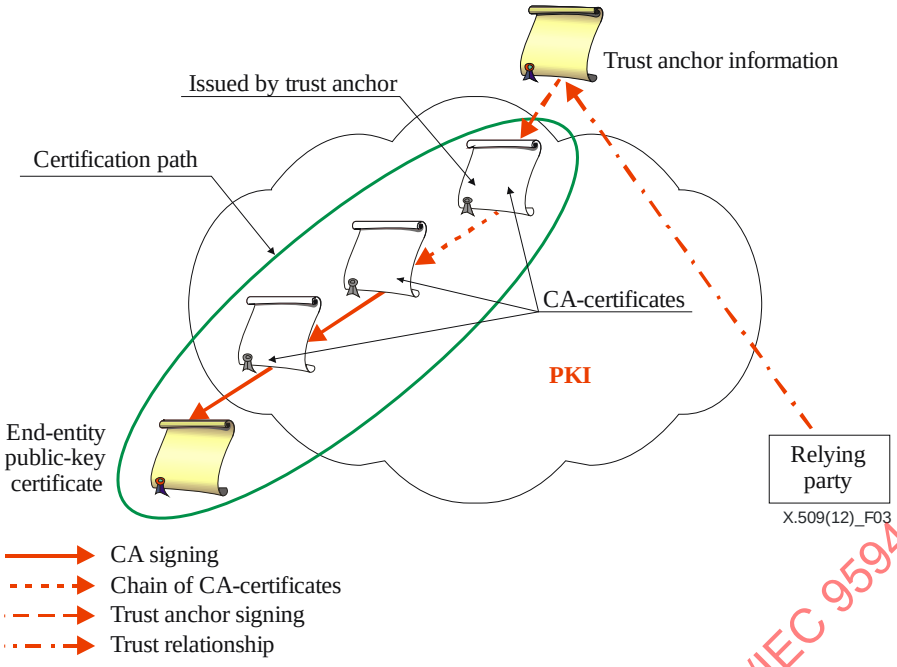


Figure 3 – Certification path

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 the end-entity and a trust anchor recognized by the relying party.

Trust suffers dilution as certification paths grow in length. The **basicConstraints** extension (see clause 8.4.2.1) 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 8.2.2.6) and **nameConstraints** (see clause 8.4.2.2). It is the responsibility of the relying party to check that the restrictions are observed.

A user may obtain one or more certificates from one or more Certification Authorities. Each certificate bears the name of the CA which issued it. The following ASN.1 data types can be used to represent certificates and a certification path:

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

CertificationPath  ::=  SEQUENCE {
     userCertificate           Certificate,
     theCACertificates              SEQUENCE OF CertificatePair OPTIONAL}
```

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

The **CACertificates** 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.

The **CertificatePair** data type is defined in clause 11.2.3. The **issuedToThisCA** component of the **CertificatePair** data type shall be present to ensure an unbroken certification path.

NOTE – The `CertificationPath` data type had already been defined by the first edition of this Directory Specification before the concept of certification path was fully developed. The order of elements in a `CertificationPath` instance appears to be 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 electronic signature. It is recommended that new applications use the `PkiPath` data type.

In addition, the following ASN.1 data type can be used to represent the forward certification path. This component contains the certification path which can point back to the originator.

```
ForwardCertificationPath ::= SEQUENCE OF CrossCertificates

CrossCertificates ::= SET OF Certificate
```

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

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 certificate is the issuer of the second certificate, etc.

Each public-key certificate in a certification path shall be unique. No public-key certificate may appear more than once in a value of the **theCACertificates** component of **CertificationPath** or in a value of **Certificate** in the **CrossCertificates** component of **ForwardCertificationPath** or a value of **Certificate** in **PkiPath**.

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 framework. 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 would be to use a "Smart Card". This would hold the private and (optionally) public keys of the user, the user's 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. The exact method chosen for storing such data, however, is beyond the scope of this Directory Specification.

Three ways in which a user's key pair may be produced are:

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

b) The key pair is generated by a third party. The third party shall release the private key to the user 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 user, and shall be subject to 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 unique distinguished name of the subject it describes. Thus:

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

b) a CA shall not issue certificates for two different subjects with the same 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 7.8 b) or of 7.8 c) is employed, the subject's private key shall be transferred to the user in a secure manner.

c) If the means of generation of key pairs of 7.8 a) or of 7.8 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 user 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 DSA or LDAP server. 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 entity's entry.

7.10 Certificate revocation list

The authority that issues certificates (public-key or attribute) also has the responsibility to indicate the validity of the certificates that it issues. Generally, certificates are subject to possible subsequent revocation. This revocation and a notification of the revocation may be done directly by the same authority that issued the certificate, or indirectly by

another authority duly authorized by the authority that issued the certificate. An authority that issues certificates is required to state, possibly through a published statement of their practices, through the certificates themselves, or through some other identified means, whether:

- the certificates cannot be revoked; or
- the certificates may be revoked by the same certificate-issuing authority directly; or
- the certificate-issuing authority authorizes a different entity to perform revocation.

Authorities that do revoke certificates are required to state, through similar means, what mechanism(s) can be used by relying parties to obtain revocation status information about certificates issued by that authority. This Directory Specification defines a Certificate Revocation List (CRL) 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 2560[1)]. Using this protocol, a relying party (client) requests the revocation status of a certificate from an OCSP server. The server may use CRLs, or other mechanisms to check the status of the certificate and respond to the client accordingly. If OCSP can be used by relying parties to check the status of a certificate, IETF RFC 5280 contains a certificate extension (Authority Info Access) that would be included in such certificates and would provide sufficient information to access an appropriate OCSP server. Relying parties check revocation status information as appropriate, for all certificates considered during the path processing procedure described in clause 10 and the delegation path processing procedure described in clause 16 to validate a 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 certificate/CRL verification. The **cRLDistributionPoints** extension can be used for this purpose. The **cRLIssuer** field of this extension would be populated with the name(s) of any entities, other than the certificate issuer itself, that have been authorized to issue CRLs concerning the revocation status of the certificate in question.

Certificates, including public-key certificates, as well as attribute 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 certificates to supersede expired/expiring certificates. Revocation notice date is the date/time that a revocation notice for a 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** field. Revocation date is the date/time the CA actually revoked the 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 certificate should otherwise be considered invalid. This date may be earlier than the revocation date. In the CRL, invalidity date is the value contained in the **invalidityDate** entry extension.

Two related points are:

- Validity of 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 authority from having to install and distribute a large number of certificates that may run out at the same expiration date.
- Expired certificates will normally be removed from the Directory. It is a matter for the security policy and responsibility of the authority to keep old certificates for a period if a non-repudiation of data service is provided.

Certificates may be revoked prior to their expiration time, e.g., if the user's private key is assumed to be compromised, the user 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 CA, and a new certificate shall be made available, if appropriate. The CA may then inform the holder of the certificate about its revocation by an offline procedure.

An authority that issues and subsequently revokes certificates:

a) may be required to maintain an audit record of its revocation events for all certificate types issued by that authority (e.g., public-key certificates, attribute certificates issued to end-entities, as well as other authorities);

b) shall provide revocation status information to relying parties using CRLs, Online Certificate Status Protocol (OCSP) 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 certificates are empty;

---

1) IETF RFC 2560, *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol (OCSP)*, June 1999.

d) if using only partitioned CRLs, it shall issue a full set of partitioned CRLs covering the complete set of 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 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 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 Directory Specification (e.g., locally configured).

The maintenance of Directory entries affected by the authority'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 certificate with a new one. The latter shall then be used to authenticate the user to the Directory.

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

- **certificateRevocationList**;
- **authorityRevocationList**;
- **deltaRevocationList**;
- **attributeCertificateRevocationList**; and
- **attributeAuthorityRevocationList**.

```
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** field shall indicate the version of the encoded revocation list. If the **extensions** component flagged as critical is present in the revocation list, the version shall be v2. If no **extensions** component flagged as critical is present in the revocation list, the version shall either be absent or present as v2.

The **signature** field shall contain the algorithm identifier for the algorithm used by the authority 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 1 – This field is redundant.

The **issuer** field shall identify the entity that has signed and issued the revocation list.

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

The **nextUpdate** field, if 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** field shall identify certificates that have been revoked. The revoked certificates are identified by their serial numbers. If none of the 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** field, if present, shall contain one or more CRL extensions.

NOTE 2 – The checking of the entire list of 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 authority, e.g., in that authority's policy.

NOTE 3 – If a non-repudiation of data service is dependent on keys provided by the authority, the service should ensure that all relevant keys of the authority (revoked or expired) and the time stamped revocation lists are archived and certified by a current authority.

NOTE 4 – If any extensions included in a CertificateList are defined as critical, the version element of the CertificateList shall be present. If no extensions defined as critical are included, the version element may be absent. If version is absent, this may permit an implementation that only supports version 1 CRLs still to use the CRL if in its examination of the revokedCertificates sequence in the CRL, it does not encounter an extension. An implementation that supports version 2 (or greater) CRLs, in the absence of version, may also be able to optimize its processing if it can determine early in processing that no critical extensions are present in the CRL.

When an implementation processing a CRL encounters the serial number of the certificate of interest in a CRL entry, but does not recognize a critical extension in the **crlEntryExtensions** field from that CRL entry, that CRL cannot be used to determine the status of the certificate. When an implementation does not recognize a critical extension in the **crlExtensions** field, that CRL cannot be used to determine the status of the certificate, regardless of whether the serial number of the 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 Directory Specification, such as seeking revocation status information from other sources.

Certificates for which revocation status cannot be determined should not be considered valid 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 certificates, or an entry may represent a range of certificates), then either that extension or a related extension shall be indicated as critical in the **crlExtensions** field. Therefore, a critical extension in the **crlEntryExtensions** field of an entry shall affect only the certificate specified in that entry, unless there is a related critical extension in the **crlExtensions** field 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 – Standard extensions for CRLs are defined in clause 8 of this Directory Specification.

If unknown elements appear within the extension, and the extension is not marked 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.

7.11 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 certificates supporting the digital signature;
– the appropriate revocation information published and available at the time of the disputed event; and,
– any 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 Directory 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 certificate has expired shall not preclude its use for revalidating signatures created during the validity period of that certificate. A certificate that has been revoked may be used if it can be determined that the 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 Public-key certificate and CRL extensions

The certificate extensions defined in this clause are for use with public-key certificates, unless otherwise stated. Extensions for use with attribute certificates are defined in clause 15. CRL extensions defined in this clause may be used in CRLs, CARLs and also for ACRLs and AARLs defined in clause 17.

This clause specifies extensions in the following areas:

a) Key and policy information: These 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 certificate policy.

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

c) Certification path constraints: These certificate extensions allow constraint specifications to be included in CA-certificates, i.e., 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 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 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 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 certificate or CRL is at the option of the authority issuing that certificate or CRL.

In a certificate or CRL, an extension is flagged as being either critical or non-critical. If an extension is flagged critical and a relying party does not recognize the extension field type or does not implement the semantics of the extension, then that relying party shall consider the certificate invalid. If an extension is flagged non-critical, a relying party that does not recognize or implement that extension type may process the remainder of the certificate ignoring the extension. If an extension is flagged non-critical, a relying party that does recognize the extension, shall process the extension. Extension type definitions in this Directory Specification indicate if the extension is always critical, always non-critical, or if criticality can be decided by the certificate or CRL issuer. The reason for requiring some extensions to be always 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 certificate in order for that 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 certificate policy identifier in the certificate policies extension with that extension being flagged critical.

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

8.1 Policy handling

8.1.1 Certificate policy

This framework contains three types of entity: the relying party, the CA and the certificate subject (or end-entity). Each entity operates under obligations to the other two entities and, in return, enjoys limited warranties offered by them. These obligations and warranties are defined in a certificate policy. A certificate policy is a document (usually in plain-

language). It may be referenced by an object identifier, which may be included in the certificate policies extension of the certificate issued by the CA, to the end-entity and upon which the relying party relies. A certificate may be issued in accordance with one or more than one policy. The definition of the policy and assignment of the identifier is performed by a policy authority. The set of policies administered by a policy authority is called a policy domain. All certificates are issued in accordance with a policy, even if the policy is neither recorded anywhere nor referenced in the certificate. This Directory 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 an authority public key and using it as trust anchor information, or by relying on a certificate that includes the associated policy identifier. The CA may be bound to its obligations under the policy by the act of issuing a certificate that includes the associated policy identifier. The end-entity may be bound to its obligations under the policy by the act of requesting and accepting a 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.

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 the policy. However, if explicitly stated in the policy, relying parties may accept the CA's assurances that its end-entities 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 Directory Specification.

A CA may place limitations on the use of its certificates, in order to control the risk that it assumes as a result of issuing certificates. For instance, it may restrict the community of relying parties, the purposes for which they may use its 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 end-entities. These matters should be defined in the certificate policy.

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

8.1.2 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 certificate subject, the subject CA and the intermediate CA are defined by the certificate policy identified in the end-entity's 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 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 **`requireExplicitPolicy`** component of the **`policyConstraints`** extension in the cross-certificate.

8.1.3 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 certificates to end-entities, 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 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.

8.1.4 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 user; 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 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.

8.1.5 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 certificate, so that it can be communicated to, and stored as trust anchor information by, its certificate-using systems;

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 certificates in a path, and shall be processed as end 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 an authority uses the same key to sign certificates and CRLs, a single self-issued certificate of category a) shall be used. If an authority uses a different key to sign CRLs than that used to sign certificates, the authority may choose to issue two self-issued certificates of category a), one for each of the keys. In this situation, relying parties would need

access to both self-issued certificates to establish separate trust anchors for certificates and CRLs signed by that authority. Alternatively, an authority may issue one self-issued certificate of category a) for certificate signing and one self-issued certificate of category b) for CRL signing. In this situation, relying parties use the key certified in the certificate of category a) as their single trust anchor for both certificates and CRLs signed by that authority. In this case, if the self-issued certificate of category b) were to be used to verify signatures on CRLs, there is no means defined in this standard to check the validity of that 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 Directory Specification.

8.2 Key and policy information extensions

8.2.1 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 field to convey an identifier of the public key to be used to verify the certificate signature. A relying party can use such identifiers in finding the correct CA-certificate for validating the certificate issuer's public key.

b) In general, a certificate subject has different public keys and, correspondingly, different certificates for different purposes, e.g., digital signature and encipherment key agreement. A certificate field is needed to assist a relying party in selecting the correct 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 certificate field 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 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 digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key. The validity period of the 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 certificates may be used in environments where multiple certificate policies apply, provision needs to be made for including certificate policy information in 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 certificate issuer 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 certificates defined in this Directory Specification needs to be able to determine in advance the algorithms supported by other users.

8.2.2 Public-key certificate and CRL extension fields

The following extension fields 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.

These extension fields shall be used only as certificate extensions, except for authority key identifier which may also be used as a CRL extension. Unless otherwise noted, these extensions may be used in both CA-certificates and end-entity certificates.

8.2.2.1 Authority key identifier extension

This field, which may be used as either a certificate extension or CRL extension, identifies the public key to be used to verify the signature on this certificate or CRL. It enables distinct keys used by the same CA to be distinguished (e.g., as key updating occurs). This field 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 certificate for the key (giving certificate issuer in the **authorityCertIssuer** component and certificate serial number in the **authorityCertSerialNumber** component), or by both explicit key identifier and identification of a certificate for the key. If both forms of identification are used then the 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 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 8.3.2.1 for details of the **GeneralNames** type.)

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

This extension is always non-critical.

8.2.2.2 Subject key identifier extension

This field 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 field 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 is always non-critical.

8.2.2.3 Key usage extension

This field identifies the intended usage for which the 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 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 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 Directory Specification, but it is anticipated that multiple methods will exist. Although not recommended, it is possible to use the content of the certificate, e.g., certificate policy, to signal the intent of the signing. However, since that signal was made when the 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 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 I.

This field is defined as follows:

```
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) }
```

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 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.

   Note that 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 Directory 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) above;

e) **keyAgreement**: for use as a public key agreement key;

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

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

Even though a digital signature was verified using a 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 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 certificate issuer, be either critical or non-critical.

If the extension is flagged critical or if the extension is flagged non-critical but the relying party recognizes it, then the certificate shall be used only for a purpose for which the corresponding key usage bit is set to one. If the extension is flagged 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.

8.2.2.4 Extended key usage extension

This field 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 field. This field 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 certificate that contains **KeyPurposeId** object identifiers for extended key usages that may be required by certificate-using applications, without restricting the 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 either critical or non-critical.

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

If the extension is flagged non-critical, then it indicates the intended purpose or purposes of the key, and may be used in finding the correct key/certificate of an entity that has multiple keys/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 certificate shall be used only for one of the purposes indicated. (Using applications may nevertheless require that a particular purpose be indicated in order for the certificate to be acceptable to that application.)

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

This Directory Specification defines the following key purpose that can be included in the extended key usage extension. Other purposes that can also be included are defined in other specifications, such as IETF RFC 5280.

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

8.2.2.5 Private key usage period extension

This field indicates the period of use of the private key corresponding to the certified public key. It is applicable only for digital signature keys. This field 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 could 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. The **notAfter** component indicates the latest date and time at which the private key could 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.

This extension is always 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 certificate validity period. With digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.

NOTE 2 – If the verifier of a digital signature wants to check that the certificate has not been revoked, for example, due to key compromise, up to the time of verification, then a valid certificate will still exist for the public key at verification time. After the certificate(s) for a public key have expired, a signature verifier cannot rely on compromises being notified via CRLs.

8.2.2.6 Certificate policies extension

This field lists certificate policies, recognized by the issuing CA, that apply to the 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 10. 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 certificate using application 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 certificate indicates the certificate policies for which this certificate is valid. The presence of this extension in a certificate issued by one CA to another CA indicates the certificate policies for which certification paths containing this certificate may be valid. This field 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** type identifies and conveys qualifier information for one certificate policy. The component **policyIdentifier** contains an identifier of a certificate policy and the component **policyQualifiers** contains policy qualifier values for that element.

This extension may, at the option of the certificate issuer, be either critical or non-critical.

If the extension is flagged critical, it indicates that the 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 non-critical, use of this extension does not necessarily constrain use of the certificate to the policies listed. However, a relying party may require a particular policy to be present in order to use the certificate (see clause 10). 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 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 Directory Specification. No object identifiers will be assigned in this Directory 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 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.

8.2.2.7 Policy mappings extension

This field, which shall be used in CA-certificates only, allows a certificate issuer to indicate that, for the purposes of the user of a certification path containing this 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 field 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 certificate issuer, be either critical or non-critical. It is recommended that it be 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.

8.3 Subject and issuer information extensions

8.3.1 Requirements

The following requirements relate to certificate subject and certificate issuer attributes:

a) Certificates need to be usable by applications that employ a variety of name forms, including Internet electronic mail names, Internet domain names, 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 certificate subject or a certificate or CRL issuer.

b) A relying party may need securely to know certain identifying information about a subject in order to have confidence that the subject is indeed the person or thing intended. For example, information such as postal address, position in a corporation, or a picture image may be required. Such information may be conveniently represented as directory attributes, but these attributes are not necessarily part of the distinguished name. A certificate field is therefore needed for conveying additional directory attributes beyond those in the distinguished name.

8.3.2 Certificate and CRL extension fields

The following extension fields are defined:

a) Subject alternative name;

b) Issuer alternative name;

c) Subject directory attributes.

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

8.3.2.1 Subject alternative name extension

This field contains one or more alternative names, using any of a variety of name forms, for the entity that is bound by the CA to the certified public key. This field 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 is a name of any form defined as an instance of the **OTHER-NAME** information object class;

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

– the **dNSName** alternative is an Internet domain name defined in accordance with IETF RFC 1035;

– the **x400Address** alternative is an O/R address defined in accordance with Rec. ITU-T X.411 | ISO/IEC 10021-4;

- the **directoryName** alternative is a distinguished name defined in accordance with Rec. ITU-T X.501 | ISO/IEC 9594-2;
- the **ediPartyName** alternative is a name of a form agreed between communicating Electronic Data Interchange partners; the **nameAssigner** component identifies an authority that assigns unique values of names in the **partyName** component;
- the **uniformResourceIdentifier** alternative is a Uniform Resource Identifier for the worldwide web defined in accordance with IETF RFC 1630;
- the **iPAddress** alternative is an Internet Protocol address defined in accordance with IETF RFC 791, represented as a binary string.
- the **registeredID** alternative is 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 the **GeneralName** type, there shall be a name registration system that ensures that any name used unambiguously identifies one entity to both the issuing CA and relying parties.

This extension may, at the option of the certificate issuer, be either critical or non-critical. An implementation which supports this extension is not required to be able to process all name forms. If the extension is flagged critical, at least one of the name forms that is present shall be recognized and processed, otherwise the 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 field of the public-key certificate contains a distinguished name that unambiguously identifies the subject, this field be flagged non-critical.

NOTE 1 – Use of the TYPE-IDENTIFIER class is described in annexes A and C of Rec. ITU-T X.681 | ISO/IEC 8824-2.

NOTE 2 – If this extension field is present and is flagged critical, the **subject** field 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).

8.3.2.2 Issuer alternative name extension

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

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

This extension may, at the option of the certificate or CRL issuer, be either critical or non-critical. An implementation which supports this extension is not required to be able to process all name forms. If the extension is flagged critical, at least one of the name forms that are present shall be recognized and processed, otherwise the 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 field of the certificate or CRL contains a distinguished name that unambiguously identifies the issuing authority, this field be flagged non-critical.

8.3.2.3 Subject directory attributes extension

This field conveys any desired Directory attributes associated with the subject of the certificate. This field 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 certificate issuer, be either critical or non-critical. A certificate-using system processing this extension is not required to understand all attribute types included in the extension. If the extension is flagged critical, at least one of the attribute types contained in the extension shall be understood for the certificate to be accepted. If the extension is flagged critical and none of the contained attribute types is understood, the certificate shall be rejected.

If this extension is present in a public-key certificate, some of the extensions defined in clause 15 may also be present.

8.4 Certification path constraint extensions

8.4.1 Requirements

For certification path processing:

a) End-entity public-key certificates need to be distinguishable from CA-certificates to protect against end-entities establishing themselves as CAs without authorization. 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 certificate or no more than two more 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 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 the local 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) Complete flexibility in trust models is required. A strict hierarchical model which is adequate for a single organization is not adequate when considering the needs of multiple interconnected enterprises. Flexibility is required in the selection of the first trusted CA in a certification path. In particular, it should be possible to require that the certification path start in the local security domain of the public-key user system.

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

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

j) 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 Directory 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.

k) A CA needs to be able to prevent the special value any-policy from being considered a valid policy in subsequent certificates in a certification path.

8.4.2 Certificate extension fields

The following extension fields are defined:

a) **`basicConstraints`**;

b) **`nameConstraints`**;

c) **`policyConstraints`**; and

d) **`inhibitAnyPolicy`**.

These extension fields shall be used only as 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 G.

8.4.2.1 Basic constraints extension

This field indicates if the subject may act as a CA, with the certified public key being used to verify certificate signatures. If so, a certification path length constraint may also be specified. This field 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** 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 issue public-key certificates only to end-entities and not to further CAs. If no **pathLenConstraint** field 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 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 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 may, at the option of the issuing CA, be either critical or non-critical. It is recommended that it be flagged critical, otherwise, an entity which is not authorized to be a CA may issue certificates and a relying party may unwittingly use such a certificate.

If this extension is present and is flagged critical, or is flagged non-critical but is recognized by the relying party, then:

- if the value of **cA** is not set to **TRUE** then the certified public key shall not be used to verify a certificate signature;
- if the value of **cA** 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, or is flagged non-critical and is not recognized by a relying party, then the public-key certificate is to be considered an end-entity public-key certificate and cannot be used to verify certificate signatures.

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

8.4.2.2 Name constraints extension

This field, 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 certificates in a certification path must 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 registeredID name forms, then in general it is not possible to constrain every possible name form of subject names with this extension.

This field 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.

If present, the **permittedSubtrees** component specifies one or more subtrees, for one or more name forms, within which subject names in acceptable certificates shall be contained. If present, the **excludedSubtrees** component specifies one or more subtrees for one or more name forms within which subject names in acceptable certificates shall not be contained. Subject names that are compared against specified subtrees include those present in both the **subject** field and the **subjectAltNames** extension of a 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** field 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** field 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 comprise the constrained name space for the name form. All subject names, in 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 certificate to be acceptable.

permittedSubtrees, if present, specifies the subtrees within which all the subject names that are of a constrained name form shall lie, for the certificate to be acceptable. If **excludedSubtrees** is present, any 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 certificate is constrained by this extension, the certificate is acceptable.

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

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 fields.

The **directoryName** name form satisfies this requirement; when using this name form a naming subtree corresponds to a DIT subtree. A **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 DN in **base**, is identical to the initial **SEQUENCE** of the same number of **RDN**s which forms the first part of the DN of the subject (in the **subject** field or **directoryName** of **subjectAltNames** extension) of the **certificate**. The **DN** of the subject of the **certificate** may have additional trailing **RDN**s in its sequence that do not appear in the DN in **base**. The **distinguishedNameMatch** matching rule is used to compare the value of **base** with the initial sequence of **RDN**s in the DN of the subject of the certificate.

Conformant implementations are not required to recognize all possible name forms. If the extension is flagged as being critical and a certificate-using implementation does not recognize a name form used in any **base** component, the

certificate shall be handled as if an unrecognized critical extension had been encountered. If the extension is flagged as being non-critical and a certificate-using implementation does not recognize a name form used in any **base** component, then that subtree may be ignored.

When testing 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 certificate issuer, be either critical or non-critical. It is recommended that it be flagged as critical; otherwise, a relying party may not check that subsequent 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 being critical, then a relying party shall check that the certification path being processed is consistent with the value in this extension.

Annex G contains examples of use of the name constraints extension.

8.4.2.3 Policy constraints extension

This field specifies constraints which may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path. This field 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 **inhibitPolicyMapping** components shall be present.

If the **requireExplicitPolicy** component 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 a certificate policy required by the user of the certification path, the identifier of a policy which has been declared equivalent to one of these policies 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 issuer of a subsequent certificate in the certification path (as indicated by a non-zero value).

If the **inhibitPolicyMapping** component is present, it indicates that, in all 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 certificate containing this extension (if the value of **inhibitPolicyMapping** is 0) or a CA which is the subject of a subsequent certificate in the certification path (as indicated by a non-zero value).

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

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

8.4.2.4 Inhibit any policy extension

This field specifies a constraint that indicates *any-policy* is not considered an explicit match for other certificate policies for all non-self-issued certificates in the certification path starting with a nominated CA. The nominated CA is either the subject CA of the certificate containing this extension (if the value of **inhibitAnyPolicy** is 0) or a CA which is the subject of a subsequent 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 certificate issuer, be either critical or non-critical. It is recommended that it be critical, otherwise a relying party may not correctly interpret the stipulation of the issuing CA.

8.5 Basic CRL extensions

8.5.1 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 8.6) and be able to detect a missing CRL in the sequence. CRL sequence numbers are therefore required.

b) Some CRL users 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 revocation.

c) There is a requirement for an authority to be able to temporarily suspend validity of a 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 certificate of an entity pending an audit or investigation.

d) A CRL contains, for each 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 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.

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

e) 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.

f) 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.

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

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

8.5.2 CRL extension fields

The following extension fields are defined:

a) CRL number;

b) CRL scope;

c) Status referral;

d) CRL stream identifier;

e) Ordered list;

f) Delta information;

g) To be revoked;

h) Revoked Group of certificates; and

i) Expired certificates on CRL.

8.5.2.1 CRL number extension

This CRL extension field conveys a monotonically increasing sequence number for each CRL issued by a given CRL issuer through a given authority directory attribute or CRL distribution point. It allows a CRL user to detect whether CRLs issued prior to the one being processed were also seen and processed. This field is defined as follows:

```
cRLNumber EXTENSION ::= {
```

```
  SYNTAX          CRLNumber
  IDENTIFIED BY   id-ce-cRLNumber }

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

This extension is always non-critical.

8.5.2.2 CRL scope extension

NOTE – Use of the CRL scope extension is deprecated.

The scope of a CRL is indicated within that CRL using the following CRL extension. In order to prevent a CRL substitution attack against an application that does not support the scope extension, the scope extension, if present, shall be marked critical.

This extension may be used to provide scope statements of various CRL types including:

- simple CRLs that provide revocation information about certificates issued by a single authority;
- indirect CRLs that provide revocation information about certificates issued by multiple authorities;
- delta-CRLs that update previously issued revocation information;
- indirect delta-CRLs that provide revocation information that updates multiple base CRLs issued by a single authority or by multiple authorities.

```
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,
  ... }
```

If the CRL is an indirect CRL that provides revocation status information for multiple authorities, the extension will include multiple **PerAuthorityScope** constructs, one or more for each of the authorities for which revocation information is included. Each instance of **PerAuthorityScope** that relates to an authority other than that issuing this CRL shall contain the **authorityName** component. If the CRL is a dCRL that provides delta revocation status information for multiple base CRLs issued by a single authority, the extension will include multiple **PerAuthorityScope** constructs, one for each of the base CRLs for which this dCRL provides updates. Even though there would be multiple instances of the **PerAuthorityScope** construct, the value of the **authorityName** component, if present, would be the same for all instances.

If the CRL is an indirect dCRL that provides delta revocation status information for multiple base CRLs issued by multiple authorities, the extension will include multiple **PerAuthorityScope** constructs, one for each of the base

CRLs for which this dCRL provides updates. Each instance of **PerAuthorityScope** that relates to an authority other than that issuing this indirect dCRL shall include the **authorityName** component.

For each instance of **PerAuthorityScope** present in the extension, the fields are used as follows. Note that in the case of indirect CRLs and indirect dCRLs, each instance of **PerAuthorityScope** may contain different combinations of these fields and different values.

The **authorityName** field, if present, identifies the authority that issued the certificates for which revocation information is provided. If **authorityName** is omitted, it defaults to the CRL issuer name.

The **distributionPoint** field, if present, is used as described in the **issuingDistributionPoint** extension.

The **onlyContains** field, if present, indicates the type(s) of certificates for which the CRL contains revocation status information. If this field is absent, the CRL contains information about all certificate types.

The **onlySomeReasons** field, if present, is used as described in the **issuingDistributionPoint** extension.

The **serialNumberRange** element, if present, is used as follows. When a modulus value is present, the serial number is reduced modulo the given value before checking for presence in the range. Then, a certificate with a (reduced) serial number is considered to be within the scope of the CRL if it is:

- equal to or greater than **startingNumber**, and less than **endingNumber**, where both are present; or
- equal to or greater than **startingNumber**, when **endingNumber** is not present; or
- less than **endingNumber** when **startingNumber** is not present.

The **subjectKeyIdRange** element, if present, is interpreted the same as **serialNumberRange**, except that the number used is the value in the certificate's **subjectKeyIdentifier** extension. The DER encoding of the **BIT STRING** (omitting the tag, length and unused bits octet) is to be regarded as the value of the DER encoding of an **INTEGER**. If bit 0 of the **BIT STRING** is set, then an additional zero octet should be prepended to ensure the resulting encoding represents a positive **INTEGER**. e.g.:

03 02 01 f 7 (represents bits 0-6 set)

maps to

02 02 00 f 7 (i.e., decimal 247)

The **nameSubtrees** field, if present, uses the same conventions for name forms as specified in the **nameConstraints** extension.

The **baseRevocationInfo** field, if present, indicates that the CRL is a dCRL with respect to the certificates covered by that **PerAuthorityScope** construct. Use of the **crlScope** extension to identify a CRL as a dCRL differs from use of the **deltaCRLIdentifier** extension in the following way. In the **crlScope** case, the information in the **baseRevocationInfo** component indicates the point in time from which the CRL containing this extension provides updates. Although this is done by referencing a CRL, the referenced CRL may or may not be one that is complete for the applicable scope, whereas the **deltaCRLIdentifier** extension references an issued CRL that is complete for the applicable scope. However, the updated information provided in a dCRL containing the **crlScope** extension are updates to the revocation information that is complete for the applicable scope regardless of whether or not the CRL referenced in **baseRevocationInfo** was actually issued as one that is complete for that same scope. This mechanism provides more flexibility than the **deltaCRLIndicator** extension since users can be constructing full CRLs locally and be constructing based on time rather than issuance of base CRLs that are complete for the applicable scope. In both cases, a dCRL always provides updates to revocation status for certificates within a given scope since a specific point in time. However, in the **deltaCRLIndicator** case, that point in time shall be one for which a CRL that is complete for that scope was issued and referenced. In the **crlScope** case, that point in time may be one for which the referenced CRL that was issued may or may not be one that is complete for that scope.

Depending on the policy of the responsible authority, several dCRLs may be published before a new base CRL is published. dCRLs containing the **crlScope** extension to reference their building point need not necessarily reference the **cRLNumber** of the most recently issued base CRL in the **BaseRevocationInfo** field. However, the **cRLNumber** referenced in the **BaseRevocationInfo** field of a dCRL shall be less than or equal to the **cRLNumber** of the most recently issued CRL that is complete for the applicable scope.

Note that the **issuingDistributionPoint** extension and **crlScope** extension can conflict with each other and are not intended to be used together. However, if the CRL contains both an **issuingDistributionPoint** extension and a **crlScope** extension, then a public-key certificate falls within the scope of the CRL if and only if it meets the criteria of both extensions. If the CRL contains an **AAissuingDistributionPoint** extension, but does not contain an **issuingDistributionPoint** or **crlScope** extension, then the scope does not include public-key certificates. If the

CRL does not contain an **issuingDistributionPoint**, **AAissuingDistributionPoint**, or **crlScope** extension, then the scope is the entire scope of the authority, and the CRL may be used for any certificate from that authority. Similarly, the **AAissuingDistributionPoint** extension and **crlScope** extension can conflict with each other and are not intended to be used together. However, if the CRL contains both an **AAissuingDistributionPoint** extension and a **crlScope** extension, then an attribute certificate falls within the scope of the CRL if and only if it meets the criteria of both extensions. If the CRL contains an **issuingDistributionPoint** extension, but does not contain an **AAissuingDistributionPoint** or **crlScope** extension, then the scope does not include attribute certificates. If the CRL does not contain an **issuingDistributionPoint**, **AAissuingDistributionPoint**, or **crlScope** extension, then the scope is the entire scope of the authority, and the CRL may be used for any certificate from that authority.

When a relying party uses a CRL that contains a **crlScope** extension to check the status of a certificate, it should check that the certificate and reason codes of interest fall within the scope of the CRL as defined by the **crlScope** extension, as follows:

a) The relying party shall check that the certificate falls within the scope indicated by the intersection of the **serialNumberRange**, **subjectKeyIdRange**, and **nameSubtrees** scopes, and is consistent with **distributionPoint**, and **onlyContains** if present, for the relevant **PerAuthorityScope** construct.

b) If the CRL contains an **onlySomeReasons** component in the **crlScope** extension, then the relying party shall check that the reason codes covered by this CRL are adequate for the purposes of the application. If not, additional CRLs may be required. Note that if the CRL contains both a **crlScope** extension and an **issuingDistributionPoint** extension, and both contain an **onlySomeReasons** component, then only those reason codes included in the **onlySomeReasons** components of both extensions are covered by this CRL.

8.5.2.3 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 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 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 certificate of interest with the scope statements and follow the pointer to the appropriate new location for revocation information relevant to that 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
```

The **issuer** component identifies the entity that signs 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** field in the most recently issued referenced CRL.

The **lastChangedCRL** component is the value of the **thisUpdate** field 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, is always flagged critical to ensure that the CRL containing this extension is not inadvertently relied on by certificate-using systems as the source of revocation status information about certificates.

If this extension is present and is recognized by a certificate-using system, 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 Directory 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 Directory Specification, to locate appropriate revocation information.

8.5.2.4 CRL stream identifier extension

The CRL stream identifier field 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)
```

This extension is always 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.

8.5.2.5 Ordered list extension

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

```
orderedList EXTENSION ::= {
```

```
  SYNTAX          OrderedListSyntax
  IDENTIFIED BY   id-ce-orderedList }

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

This extension is always non-critical.

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

8.5.2.6 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,
  ... }
```

This extension is always non-critical.

8.5.2.7 To be revoked extension

This CRL extension allows for the notification that 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 certificate revocation, the date and time at which the certificate will be revoked, and the group of certificates to be revoked. Each list can contain a single certificate serial number, a range of certificate serial numbers or a named **subtree**. These certificates may be public-key certificates or attribute certificates.

```
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 authority (CA or AA) that issued all the certificates listed in this **ToBeRevokedGroup**. If **certificateIssuer** is omitted, it defaults to the CRL issuer name.

The **reasonInfo** component, if present, identifies the reason for the certificate revocations. If present, this field indicates that all certificates identified in **ToBeRevokedGroup** will be revoked for the reason indicated in this field. If **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 certificates identified in **RevokedGroup**. This action should only be taken, after the revocation time indicated in the **revocationTime** field has passed.

The **revocationTime** component indicates the date and time at which this group of 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 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 certificates will be revoked. Once the revocation time has passed, either the CA has revoked the certificate or not. If it has revoked the certificate, future CRLs shall include this on the list of revoked certificates, at least until the certificate expires. If the CA has not revoked the certificate, but still intends to revoke it in the future, it may include the certificate in this extension on subsequent CRLs with a revised **revocationTime**. If the CA no longer intends to revoke the certificate, it may be excluded from all subsequent CRLs and the certificate shall not be considered revoked.

The **certificateGroup** component lists the set of public-key certificates to be revoked. This component identifies the certificates issued by the authority 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, if present, shall hold the serial number(s) of the certificate(s) issued by the identified certificate issuer that will be revoked at the specified time.

If the **serialNumberRange** component is present, all certificates in the range beginning with the starting serial number and ending with the ending serial number and issued by the identified certificate issuer will be revoked at the specified time.

If the **nameSubtree** component is present, all public-key certificates with a subject/holder name that is subordinate to the specified name and issued by the identified certificate issuer 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** field and **subjectAltNames** extension) or **holder** field of an attribute certificate need to be considered. For other name forms, the **subjectAltNames** extension of public-key certificates and the **holder** field of attribute certificates need to be considered. If at least one of the names associated with the subject/holder, contained in the certificate, is within the subtree specified in **nameSubtree**, that 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 critical or 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 non-critical, reducing the risk of problems with interoperability and backward compatibility.

8.5.2.8 Revoked group of certificates extension

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

```
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 authority (CA or AA) that issued all the certificates listed in this **RevokedGroup**. If **certificateIssuer** is omitted, it defaults to the CRL issuer name.

The **reasonInfo** component, if present, identifies the reason for the certificate revocations. If present, this field indicates that all certificates identified in **RevokedGroup** were revoked for the reason indicated in this field. If **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 certificates identified in **RevokedGroup**.

The **invalidityDate** component, if present, indicates the time from which all certificates identified in **RevokedGroup** should be considered invalid. This date shall be earlier than the date contained in **thisUpdate** field of the CRL. If omitted, all certificates identified in **RevokedGroup** should be considered invalid at least from the time indicated in the **thisUpdate** field of the CRL. If the status of the certificate prior to the **thisUpdate** time is critical to a certificate-using system (e.g., to determine whether a digital signature that was created prior to this CRL issuance occurred while the 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 certificate should be considered invalid.

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

If **serialNumberRange** is present, all certificates containing certificate serial numbers within the specified range, issued by the identified certificate issuer are applicable.

If **nameSubtree** is present, all certificates with a subject/holder name that is subordinate to the specified name and issued by the identified certificate issuer will be revoked at the specified time. If the **nameSubtree** contains a DN then all DNs associated with the subject of a public-key certificate (i.e., **subject** field and **subjectAltNames** extension) or **holder** field of an attribute certificate need to be considered. For other name forms, the **subjectAltNames** extension of public-key certificates and the **holder** field of attribute certificates need to be considered. If at least one of the names associated with the subject/holder, contained in the certificate, is within the subtree specified in **nameSubtree**, that 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 critical. Otherwise, a certificate-using system may incorrectly assume that certificates, identified as revoked within this extension, are not revoked. When this extension is present it may be the only indication of revoked certificates in a CRL (i.e., the **revokedCertificates** may be empty) or it may list revoked certificates that are in addition to those indicated in the **revokedCertificates** field. A revoked certificate shall not be listed both in the **revokedCertificates** field and in this extension.

8.5.2.9 Expired certificates on CRL extension

This CRL extension field indicates that the CRL includes revocation notices for expired certificates.

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

ExpiredCertsOnCRL ::= GeneralizedTime
```

This extension is always non-critical.

The scope of a CRL containing this extension is extended to include the revocation status of certificates that expired at the exact time specified in the extension or after that time. If limitations in the CRL's scope are specified (by either reason codes or by distribution points), that applies to expired certificates as well. The revocation status of a certificate shall not be updated once the certificate has expired.

8.5.3 CRL entry extension fields

The following extension fields are defined:

a) Reason code;

b) Hold instruction code; and

c) Invalidity date.

8.5.3.1 Reason code extension

This CRL entry extension field identifies a reason for the certificate revocation. The reason code may be used by applications to decide, based on local policy, how to react to posted revocations. This field 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),
  ... }
```

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

– **unspecified** can be used to revoke certificates for reasons other than the specific codes.

– **keyCompromise** is used in revoking an end-entity certificate; it indicates that it is known or suspected that the subject's private key, or other aspects of the subject validated in the 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 certificate, have been compromised.

– **affiliationChanged** indicates that the subject's name or other information in the certificate has been modified but there is no cause to suspect that the private key has been compromised.

– **superseded** indicates that the certificate has been superseded but there is no cause to suspect that the private key has been compromised.

– **cessationOfOperation** indicates that the 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 certificate (public-key or attribute certificate) was revoked because a privilege contained within that 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.

A certificate may be placed on hold by issuing a CRL entry with a reason code of **certificateHold**. The certificate hold notice may include an optional hold instruction code to convey additional information to relying parties (see clause 8.5.2.3). 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 users to reject transactions issued during the hold period;

b) it may be replaced by a (final) revocation for the same certificate, in which case the reason shall be one of the standard reasons for revocation, the revocation date shall be the date the certificate was placed on hold, and the optional instruction code extension field 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 8.6) only and indicates that an existing CRL entry should now be removed owing to 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 certificate with reason code **certificateHold**.

This extension is always non-critical.

8.5.3.2 Hold instruction code extension

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

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

HoldInstruction ::= OBJECT IDENTIFIER
```

This extension is always non-critical. No standard hold instruction codes are defined in this Directory Specification.

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

8.5.3.3 Invalidity date extension

This CRL entry extension field indicates the date at which it is known or suspected that the private key was compromised or that the 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 field is defined as follows:

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

This extension is always 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.

8.6 CRL distribution points and delta-CRL extensions

8.6.1 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 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 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 authorities. Since it already has a list of previously revoked certificates, it only needs to retrieve a list of newly revoked 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 certificates issued by one authority to different CRLs. This can be achieved by associating every certificate with a CRL distribution point which is either:
   - a Directory entry whose CRL attribute will contain a revocation entry for that 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 certificates, e.g., a certification path. This can be achieved by having one CRL issuer sign and issue CRLs containing revocations from multiple authorities.

c) There is a requirement for separate CRLs covering revoked authority certificates and revoked end-entity certificates. This facilitates the processing of certification paths as the CRL for revoked authority certificates can be expected to be very short (usually empty). The **authorityRevocationList** and **certificateRevocationList** attributes 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 certificates that have been revoked since the issuance of a base CRL.

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

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

8.6.2 CRL distribution point and delta-CRL extension fields

The following extension fields are defined:

a) CRL distribution points;

b) Issuing distribution point;

c) AAissuingDistributionPoint;

d) Certificate issuer;

e) Delta CRL indicator;

f) Base update;

g) Freshest CRL.

CRL distribution points and freshest CRL shall be used only as a certificate extension. Issuing distribution point, AA 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.

While the issuing distribution point extension and the AA issuing distribution point extension serve similar purposes, they apply to different certificates. The issuing distribution point extension applies only to public-key certificates issued to end-entities and/or CAs. The AA issuing distribution point extension applies only to attribute certificates issued to users and AAs, as well as public-key certificates issued to SOAs. If a single CRL covers certificate types that span these, then that CRL would need to include both extensions.

8.6.2.1 CRL distribution points extension

The CRL distribution points extension shall be used only as a certificate extension and may be used in authority-certificates, end-entity public-key certificates and in attribute certificates. This field identifies the CRL distribution point or points to which a relying party should refer to ascertain if the 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 authority directory entry.

This field 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) }
```

The **distributionPoint** component identifies the location from which the CRL can be obtained. If this component is absent, the distribution point name defaults to the CRL issuer name.

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 issuing distribution point 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 certificate provided requisite revocation information can be obtained from another source, e.g., another distribution point or the authority's directory entry.

The **nameRelativeToCRLIssuer** component 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 indicates 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 certificate if this certificate has been revoked, regardless of revocation reason. Otherwise, the **reasons** value indicates which revocation reasons are covered by the corresponding CRL distribution point.

The **cRLIssuer** component identifies the authority that issues and signs the CRL. If this component is absent, the CRL issuer name defaults to the certificate issuer name.

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

If this extension is flagged critical then a relying party shall not use the 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 all certificates issued by the CA include the **cRLDistributionPoints** as a critical extension, the CA is not required to also publish a full CRL at the CA entry.

If this extension is flagged non-critical and a relying party does not recognize the extension field type, then that system should only use the certificate 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 field in the CRL);
- revocation checking is not required under local policy; or
- revocation checking is accomplished by other means.

NOTE 1 – It is possible to have CRLs issued by more than one CRL issuer for the one certificate. Coordination of these CRL issuers and the issuing authority is an aspect of authority policy.

NOTE 2 – The meaning of each reason code is as defined in the Reason Code field in clause 8.5.3.1 of this Directory Specification.

8.6.2.2 Issuing distribution point extension

This CRL extension field 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 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 certificates, if the CRL issuer was not using partitioned CRLs. The limitation may be based on a subset of the certificate population or on a subset of revocation reasons. The CRL is signed by the CRL issuer's private key – CRL distribution points do not have their own key pairs. However, for a CRL distributed via the 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 issuing distribution point field, the AA issuing distribution point field, and the CRL scope field are all absent, the CRL shall contain entries for all revoked unexpired public-key certificates issued by the CRL issuer. If the

issuing distribution point field and the CRL scope field are both absent, but the AA issuing distribution point field is present, the scope of the CRL does not include public-key certificates.

After a certificate appears on a CRL, it may be deleted from a subsequent CRL after the certificate's expiry. This field 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,
  ... }
```

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 authorities that have a name different from the name of the issuer of the CRL. The particular authority 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 8.6.2.3. Consequently, a certificate using a system 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 CRLs are partitioned by reason code, and the reason code changes for a revoked certificate (causing the certificate to move from one CRL stream to another), it is necessary to continue to include the certificate on the CRL stream for the old revocation reason until the **nextUpdate** times of all CRLs, that do not list the 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 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** field is present. In that case, a name comparison would be done on the full DN, constructed by appending the value of the **nameRelativeToCRLIssuer** to the DN found in the **issuer** field of the CRL. If the names being compared are DNs (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 the 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 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;

– a CRL which contains both an **issuingDistributionPoint** extension and an **aAissuingDistributionPoint** extension with **containsUserAttributeCerts** set to **TRUE** 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 is always critical. A relying party that does not understand this extension cannot assume that the CRL 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 CRL issuers is beyond the scope of this Directory 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.

8.6.2.3 Certificate issuer extension

This CRL entry extension identifies the certificate issuer 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 certificate issuer defaults to the CRL issuer. On subsequent entries in an indirect CRL, if this extension is not present, the certificate issuer for the entry is the same as that for the preceding entry.

This field is defined as follows:

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

This extension is always critical. If an implementation ignores this extension, it cannot correctly pair CRL entries to certificate issuers.

8.6.2.4 Delta CRL indicator extension

The delta CRL indicator field 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 certificate revocation status for that same scope. This scope does not necessarily include all revocation reasons or all 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 field 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 is always 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.

8.6.2.5 Base update extension

The base update field 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 does not require this extension as the **baseThisUpdate** field of the **crlScope** extension can be used for the same purpose.

```
baseUpdateTime EXTENSION ::= {
  SYNTAX          GeneralizedTime
```

```
  IDENTIFIED BY  id-ce-baseUpdateTime }
```

This extension is always non-critical.

8.6.2.6 Freshest CRL extension

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

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

The value of type **CRLDistPointsSyntax** is as defined in the CRL distribution points extension in clause 8.6.2.1.

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

8.6.2.7 AA issuing distribution point extension

This CRL extension field identifies the CRL distribution point for attribute 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. The limitation may be based on a subset of the certificate population or on a subset of revocation reasons. The CRL is signed by the CRL issuer's private key – CRL distribution points do not have their own key pairs. However, for a CRL distributed via the 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 issuing distribution point extension, the AA issuing distribution point extension, and the CRL scope field are all absent, the CRL shall contain entries for all revoked unexpired attribute certificates issued by the CRL issuer. If the AA issuing distribution point field and the CRL scope field are both absent, but the issuing distribution point field is present, the scope of the CRL does not include attribute certificates.

After a certificate appears on a CRL, it may be deleted from a subsequent CRL after the certificate's expiry.

This field 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 CRL only contains revocations for attribute 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 attribute certificates from authorities other than the issuer of the CRL. The particular authority responsible for each entry is as indicated by the certificate issuer CRL entry extension in that entry or in accordance with the defaulting rules described in clause 8.6.2.3. 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 **containsUserAttributeCerts**, **containsAACerts**, **containsSOAPublicKeyCerts** and **onlySomeReasons** indicators, from all authorities that identify this CRL issuer in their attribute certificates.

If **containsUserAttributeCerts** is **TRUE**, the CRL contains revocations for attribute certificates issued to end entities that are not themselves AAs. If **containsAACerts** is **TRUE**, the CRL contains revocations for attribute certificates issued to subjects that are themselves AAs.

If **containsSOAPublicKeyCerts** is **TRUE**, the CRL contains revocations for public-key certificates issued to an entity that is an SOA for the purposes of privilege management (i.e., certificates that contain the **SOAIdentifier** extension). For CRLs distributed via the 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 certificate types covered by the CRL. Unless the CRL is a dCRL:

- a CRL 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;
- a CRL 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 CRL issuer entry;
- a CRL which contains an **issuingDistributionPoint** extension shall be distributed as specified in clause 8.6.2.2.

This extension is always critical. A relying party that does not understand this extension cannot assume that the CRL 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 CRL issuers is beyond the scope of this Directory Specification.

If an authority publishes a CRL with containsAACerts set to TRUE and 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 a CRL with containsSOAPublicKeyCerts set to TRUE, then the authority shall ensure that all SOA certificates covered by this CRL contain the SOAIdentifier extension.

9 Delta CRL relationship to base

A dCRL includes either a **deltaCRLIndicator** or a **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 **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 authorities. The **crlScope** extension shall be used as the means of identifying a CRL as an indirect dCRL. The **crlScope** extension shall contain one instance of the **PerAuthorityScope** data type for each base CRL for which the indirect dCRL provides updated information.

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

- A 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 certificate pending a further decision on whether to permanently revoke the certificate or reinstate it as one that is not revoked.

a) If a 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 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 certificate shall be listed with revocation reason **removeFromCRL** unless the certificate is subsequently revoked again for one of the revocation reasons covered by the dCRL, in which case the 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 certificate's revocation notice may first appear on dCRL and it is possible that the 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 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.

10 Certification path processing procedure

Certification path processing is carried out in a system which needs to use the public key of a remote end entity, e.g., a system which is verifying a digital signature generated by a remote entity. The certificate policies, basic constraints, name constraints and policy constraints extensions 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.

10.1 Path processing inputs

The inputs to the certification path processing procedure are:

a) a set of certificates comprising a certification path;

b) a trusted public key value or key identifier (if the key is stored internally to the certification path processing module), 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 field 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 certification path processing module);

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 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. For clause 10, subject names are those name values appearing in the subject field 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 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 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.

Note that 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.

10.2 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 certificate in the path; and

f) details of any policy mapping that occurred in processing the certification path.

NOTE – If validation is successful, the relying party may still choose not to use the certificate as a result of values of policy qualifiers or other information in the certificate.

10.3 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 certificates of the certification path (rows represent policies, their qualifiers and mapping history, and columns represent certificates in the certification path).

b) permitted-subtrees: A set of subtree specifications defining subtrees within which all subject names in subsequent 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 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 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 certificate in the path.

f) path depth: An integer equal to one more than the number of 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 certificates yet to skip before the constraint takes effect.

10.4 Initialization step

The procedure involves an initialization step, followed by a series of 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.

10.5 Certificate processing

Each certificate is then processed in turn, starting with the certificate signed using the input trusted public key. The last certificate is considered to be the end certificate; any other certificates are considered to be intermediate certificates.

10.5.1 Basic certificate checks

The following checks are applied to a certificate. Self-signed certificates, if encountered in the path, are ignored.

a) Check that the signature verifies, that dates are valid, that the certificate subject and certificate issuer names chain correctly, and that the certificate has not been revoked.

b) For an intermediate version 3 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 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 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 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 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 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 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 certificate of one of the name forms in the set.

10.5.2 Processing intermediate certificates

For an intermediate 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 c) above (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].

10.5.3 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 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).

10.5.4 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*.

11 PKI directory schema

This clause defines the directory schema elements used to represent PKI information in the Directory. It includes specification of relevant object classes, attributes and attribute value matching rules.

11.1 PKI directory object classes and name forms

This subclause includes the definition of object classes used to represent PKI objects in the Directory.

11.1.1 PKI user object class

The PKI user object class is used in defining entries for objects that may be the subject of public-key certificates.

```
pkiUser OBJECT-CLASS ::= {
  SUBCLASS OF  {top}
  KIND         auxiliary
  MAY CONTAIN  {userCertificate}
  ID           id-oc-pkiUser }
```

11.1.2 PKI CA object class

The PKI CA object class is used in defining entries for objects that act as certification authorities.

```
pkiCA OBJECT-CLASS ::= {
  SUBCLASS OF  {top}
  KIND         auxiliary
  MAY CONTAIN  {cACertificate |
                certificateRevocationList |
                authorityRevocationList |
                crossCertificatePair}
  ID           id-oc-pkiCA }
```

11.1.3 CRL distribution points object class and name form

The CRL Distribution Point object class is used in defining entries for object which act as CRL Distribution Points.

```
cRLDistributionPoint OBJECT-CLASS ::= {
  SUBCLASS OF   {top}
  KIND          structural
  MUST CONTAIN  {commonName}
  MAY CONTAIN   {certificateRevocationList |
                 authorityRevocationList |
                 deltaRevocationList}
  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 }
```

11.1.4 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}
  ID           id-oc-deltaCRL }
```

11.1.5 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}
  ID           id-oc-cpCps }
```

11.1.6 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}
  ID           id-oc-pkiCertPath }
```

11.2 PKI directory attributes

This subclause includes the definition of directory attributes to store PKI information elements in the Directory.

11.2.1 User certificate attribute

A user may obtain one or more public-key certificates from one or more CAs. The **userCertificate** attribute type contains the end-entity public-key certificates a user has obtained from one or more CAs.

```
userCertificate ATTRIBUTE ::= {
  WITH SYNTAX              Certificate
  EQUALITY MATCHING RULE   certificateExactMatch
  ID                       id-at-userCertificate }
```

11.2.2 CA-certificate attribute

The **cACertificate** attribute of a CA's directory entry shall be used to store self-issued certificates (if any) and certificates issued to this CA by CAs in the same realm as this CA. In the case of v3 certificates, these certificates shall include a **basicConstraints** extension with the **cA** value set to **TRUE**. The definition of realm is purely a matter of local policy.

```
cACertificate ATTRIBUTE ::= {
  WITH SYNTAX              Certificate
  EQUALITY MATCHING RULE   certificateExactMatch
  ID                       id-at-cAcertificate }
```

11.2.3 Cross-certificate pair attribute

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** elements of the **crossCertificatePair** attribute, of a CA's directory entry may contain a subset of 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 certificate shall match the subject name in the other and vice versa, and the subject public key in one certificate shall be capable of verifying the digital signature on the other 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.

In the case of v3 certificates, these shall include a **basicConstraints** extension with the **cA** value set to **TRUE**.

```
crossCertificatePair ATTRIBUTE ::= {
  WITH SYNTAX              CertificatePair
  EQUALITY MATCHING RULE   certificatePairExactMatch
  ID                       id-at-crossCertificatePair }

CertificatePair ::= SEQUENCE {
  issuedToThisCA  [0]  Certificate OPTIONAL,
  issuedByThisCA  [1]  Certificate OPTIONAL,
  ... }
  (WITH COMPONENTS { ..., issuedToThisCA PRESENT} |
   WITH COMPONENTS { ..., issuedByThisCA PRESENT})
```

11.2.4 Certificate revocation list attribute

The following attribute contains a list of revoked certificates.

```
certificateRevocationList ATTRIBUTE ::= {
  WITH SYNTAX              CertificateList
  EQUALITY MATCHING RULE   certificateListExactMatch
  ID                       id-at-certificateRevocationList }
```

11.2.5 Authority revocation list attribute

The following attribute contains a list of revoked authority certificates.

```
authorityRevocationList ATTRIBUTE ::= {
  WITH SYNTAX              CertificateList
  EQUALITY MATCHING RULE   certificateListExactMatch
  ID                       id-at-authorityRevocationList }
```

11.2.6 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
  ID                       id-at-deltaRevocationList }
```

11.2.7 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 certificates as defined in this Directory Specification. The following ASN.1 defines this (multi-valued) attribute:

```
supportedAlgorithms ATTRIBUTE ::= {
  WITH SYNTAX              SupportedAlgorithm
  EQUALITY MATCHING RULE   algorithmIdentifierMatch
  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 8.2.2.3 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.

11.2.8 Certification practice statement attribute

The **certificationPracticeStmt** attribute is used to store information about an authority'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 implementors
```

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.

11.2.9 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 11.2.8) has the following 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 the directory. The HASH stored in the 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).

11.2.10 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 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 contain certification paths that include the end-entity certificate. In this case, the end-entity is the user whose entry holds this attribute. The values of the attribute represent complete certification paths for public-key certificates issued to this user.

11.3 PKI directory matching rules

This subclause defines matching rules for use with attribute types with syntax **Certificate**, **CertificatePair**, **CertificateList**, **CertificatePolicy**, and **SupportedAlgorithm**, respectively. This subclause also defines matching rules to facilitate the selection of certificates or CRLs with specific characteristics from multi-valued attributes holding multiple certificates or CRLs. The enhanced certificate matching rule provides the ability to perform more sophisticated matching against certificates held in directory entries.

11.3.1 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.

11.3.2 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 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 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.

11.3.3 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.

11.3.4 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.

11.3.5 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 rule returns TRUE if the components in the stored attribute value match those in the presented value. If the **distributionPoint** component is present, then it shall match in at least one name form.

11.3.6 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.

11.3.7 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.

11.3.8 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.

11.3.9 PKI path match

The **pkiPathMatch** match rule compares for equality a presented value with an attribute value of type **pkiPath**. A certificate-using system may use this matching rule to select a path beginning with a certificate issued by a CA which it trusts and ending with a 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** field of the first certificate in the **SEQUENCE** in the stored value and the presented value in the **lastSubject** component matches the corresponding elements of the subject field of the last certificate in the **SEQUENCE** in the stored value. This matching rule returns FALSE if either match fails.

11.3.10 Enhanced certificate match

The enhanced certificate match rule compares a presented value with an attribute value of type **Certificate**. It selects one or more 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** fields. 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 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 certificates that form a path to an end-entity 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 DN in the **pathToName** component. A stored 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 certificate and would therefore not be a matched value for this sample assertion.

11.4 PKI directory syntax definitions

11.4.1 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.

11.4.2 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.

11.4.3 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.

11.4.4 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.

11.4.5 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.

11.4.6 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.

11.4.7 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.

11.4.8 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.

11.4.9 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.

11.4.10 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.

11.4.11 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 an extension defined explicitly for this purpose. The format of attribute certificates is defined here, including an extensibility mechanism and a set of specific certificate extensions. 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 need to be able to learn that revocation has occurred so they do not use an untrustworthy certificate. Revocation lists are one scheme that can be used to notify users of revocations. The format of revocation lists is defined in Section 2 of this Directory Specification, including an extensibility mechanism and a set of revocation list extensions. Additional extensions are defined here. In both the certificate and revocation list case, other bodies may also define additional extensions that are useful to their specific environments.

An attribute certificate-using system needs to validate a certificate prior to using that certificate for an application. Procedures for performing that validation are also defined here, including verifying the integrity of the 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 Directory Specification so that those environments that do have requirements for them can also be supported.

The Directory uses attribute certificates to provide rule-based access control to Directory information.

12 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 Directory 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 through the practices of the issuing CA. Public-key certificates may contain a **subjectDirectoryAttributes** extension that contains privileges associated with the subject of the public-key certificate. This mechanism is appropriate in situations where the authority issuing the public-key certificate (CA) is also the authority for delegating the privilege (AA) and the validity period of the privilege corresponds to the validity period of the public-key certificate. End-entities cannot act as AAs. If any of the extensions defined in clause 15 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.

12.1 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 a user; 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 Directory Specification.

The attribute certificate is defined as follows.

```
AttributeCertificate ::= SIGNED{AttributeCertificateInfo}

AttributeCertificateInfo ::= 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 }

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 **version** field shall specify the version of the attribute certificate. For attribute certificates issued in accordance with the syntax in this Directory Specification, **version** shall be **v2**.

The **holder** field shall convey the identity of the attribute certificate's holder 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 certificate (i.e., the identical key value may appear in multiple 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** field shall convey the identity of the AA that issued the certificate.

– The **issuerName** component, if present, shall identify one or more names for the issuer.

– The **baseCertificateID** component, if present, shall identify the issuer by reference to a specific public-key certificate for which this issuer is the subject.

– The **objectDigestInfo** component, if present, shall identify the issuer by providing a hash of identifying information for the issuer.

The **signature** field shall identify the cryptographic algorithm used to digitally sign the attribute certificate.

NOTE 2 – This field is redundant.

The **serialNumber** field shall be a serial number that uniquely identifies the attribute certificate within the scope of its issuer.

The **attrCertValidityPeriod** field shall convey the time period during which the attribute certificate is considered valid, expressed in **GeneralizedTime** format.

The **attributes** field 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** field 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 field is deprecated. This field was added because at one time there was some fear of the reuse of distinguished names.

The **extensions** field allows the addition of new fields to the attribute certificate.

If unknown elements appear within the extension, and the extension is not marked 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. However, as mentioned earlier, the certificate extensions defined in this section can also be placed in a public-key certificate using the **subjectDirectoryAttributes** extension.

12.2 Attribute certification paths

Just as with public-key certificates, there may be a requirement to convey an attribute certification path (e.g., within an application protocol to assert privileges). The following ASN.1 data type can be used to represent an attribute certification path:

```
AttributeCertificationPath ::= SEQUENCE {
  attributeCertificate  AttributeCertificate,
  acPath                SEQUENCE OF ACPathData OPTIONAL,
  ... }

ACPathData ::= SEQUENCE {
  certificate           [0]  Certificate OPTIONAL,
  attributeCertificate  [1]  AttributeCertificate OPTIONAL,
  ... }
```

13 Attribute Authority, SOA and Certification Authority relationship

The Attribute Authority (AA) and Certification Authority (CA) are logically (and, in many cases, physically) 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 for identity within its domain, is not automatically the source of authority 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 Source of Authority (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 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 Directory 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 individuals are issued certificates that assign various roles to them. The privileges associated with the role are implicitly assigned to such individuals. 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 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) In some environments, the same physical entity may be acting as both an AA and a CA. This dual logical role for the same physical entity is always the case when privilege is conveyed within the **`subjectDirectoryAttributes`** extension of a public-key certificate. In other environments, separate physical entities act as CAs and AAs. In the latter case, privilege is assigned using attribute certificates instead of public-key certificates.

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 Directory 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 Directory Specification. The first delegation model is one where the privilege delegator is an AA that can issue certificates delegating that privilege to others. The second model allows for an independent Delegation Service (DS) in which the entity issues certificates on behalf of another AA (that may or may not be able to issue ACs 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 AC 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 relying party will have to check that delegation was performed correctly.

Two *recognition of authority* models are described in this Directory 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)s prior to them making access control decisions for users who originate from the remote domain. No support for static RoA is provided in this Directory Specification. With dynamic RoA, the local SOA issues new supplementary policy ACs that add additional information to the current policy. Remote SOAs may also be recognized to issue supplementary policy ACs for the local PDPs. In both cases, these new supplementary policy ACs 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.

13.1 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 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 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.

13.2 Privilege in public-key certificates

In some environments, privileges are associated with the subject through the practices of a CA. Such 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 same physical entity is acting both as a CA and an AA;
- 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 certificate (in the **`subjectDirectoryAttributes`** extension) have the same delegation parameters and all extensions relevant to delegation apply equally to all privileges in the certificate.

14 PMI models

14.1 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 the 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 Directory Specification defines one simple privilege attribute type. Other examples are provided in Annex E.

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 system. Between these extremes are many shades of variation. Schema components for storing privilege policy information in the Directory are defined in this Directory 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 Directory Specification. Annex E 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 fields 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 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 Directory Specification, such as privilege asserter, privilege verifier, etc., are independent of the particular application or use.

14.1.1 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 Directory Specification to those in the access control framework is provided here, to clarify the relationship between this model and that Directory Specification.

Service request in this Directory Specification corresponds to the 'access request' defined in the access control framework.

Privilege asserter in this Directory Specification would be acting in the role of an 'initiator' in the access control framework.

Privilege verifier in this Directory 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 Directory Specification would correspond to the 'target' defined in the access control framework.

Environmental variables in this Directory Specification would correspond to the 'contextual information' in the access control framework.

Privilege policy discussed in this Directory 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.

14.1.2 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 Directory Specification to those in the non-repudiation framework is provided here, to clarify the relationship between this model and that Directory Specification.

Privilege asserter in this Directory Specification would be acting in the role of an 'evidence subject' or an 'originator' in the non-repudiation framework.

Privilege verifier in this Directory 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 Directory Specification would correspond to the 'target' defined in the non-repudiation framework.

Environmental variables in this Directory Specification would correspond to the date and time the evidence was generated or verified' in the non-repudiation framework.

Privilege policy discussed in this Directory Specification could include 'non-repudiation security policy' in the non-repudiation framework.

14.2 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 4). 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.

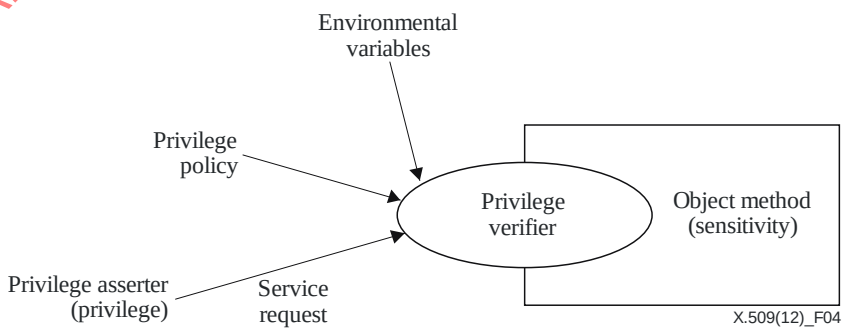


Figure 4 – Control model

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.

14.3 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 5).

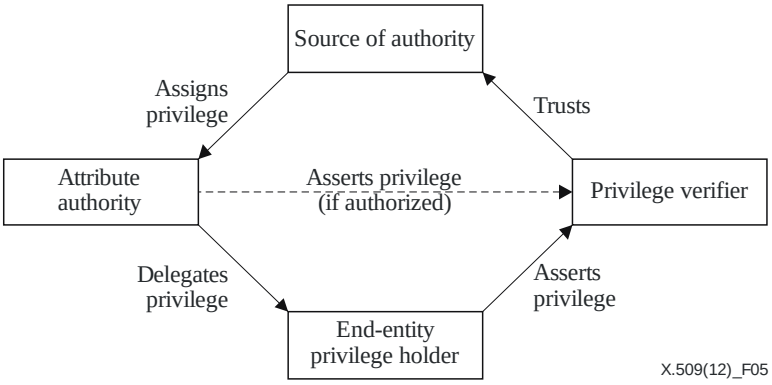


Figure 5 – Delegation model

As with environments where delegation is not used, the SOA is the initial issuer of 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 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 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 certificate validation 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 AAs may delegate privilege. End-entities cannot.

14.4 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 AC to the group.

There are two ways of identifying the members of a group who are assigned a group AC. These methods are called direct group naming and group role naming.

14.4.1 Direct group naming

In direct group naming, the **`holder`** field of the group AC 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 AC.

14.4.2 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** field of the group AC takes the **entityName** option and holds the role(s) of the group members who are being assigned the **attributes** in this group AC. The **GeneralNames** should contain a single **GeneralName** containing a **directoryName** with a single RDN, whose attribute type is the **role** attribute defined in clause 14.5.1. 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 attribute authorities to issue the role assignment certificates shall be determined through means outside this Directory 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 AC.

NOTE 2 – Where the role in the holder field is the same as the role in the **attributes** field of this group AC, this is delegation of authority from the issuer of the group AC 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.

14.5 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 the role attribute 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 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 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 certificate or outside the PMI (e.g., locally configured). If the role privileges are asserted in a role specification certificate, mechanisms for linking that certificate with the relevant role assignment certificate for the privilege asserter are provided in this Directory Specification. A role specification 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 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.

14.5.1 Role attribute

The specification of privilege attribute types is generally an application-specific issue that is outside the scope of this Directory 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 Directory Specification.

```
role ATTRIBUTE ::= {
  WITH SYNTAX  RoleSyntax
  ID           id-at-role }

RoleSyntax ::= SEQUENCE {
  roleAuthority  [0]  GeneralNames OPTIONAL,
  roleName       [1]  GeneralName,
  ... }
```

This privilege attribute may be used to populate the **attributes** field of a role assignment certificate or to populate the **holder** field of a role specification or group attribute certificate, or both.

If the role assignment certificate is a public-key certificate rather than an AC, 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** field of a role assignment certificate, the **roleAuthority**, if present, identifies the recognized authority that is responsible for issuing the role specification 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** field of that role specification certificate. If the privilege verifier has used means other than a role specification 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 Directory 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 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** field of the role specification certificate.

When the **role** attribute is used to populate the **holder** field 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** field 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** field and the **attributes** field, this is a role mapping attribute certificate.

14.6 Recognition of Authority Model

Figure 6 shows the control model for a single domain ITU-T X.509 PMI.

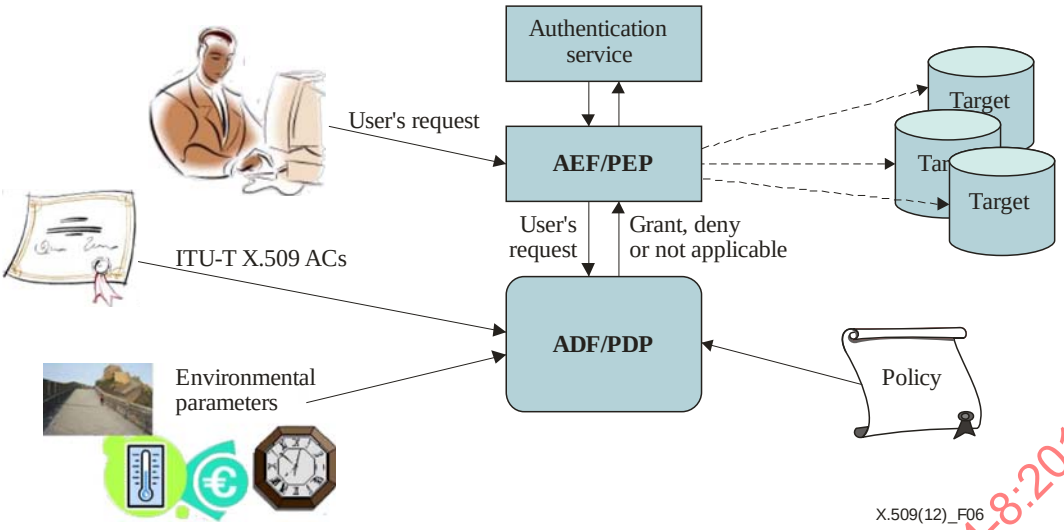


Figure 6 – The control model for a single domain PMI

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.

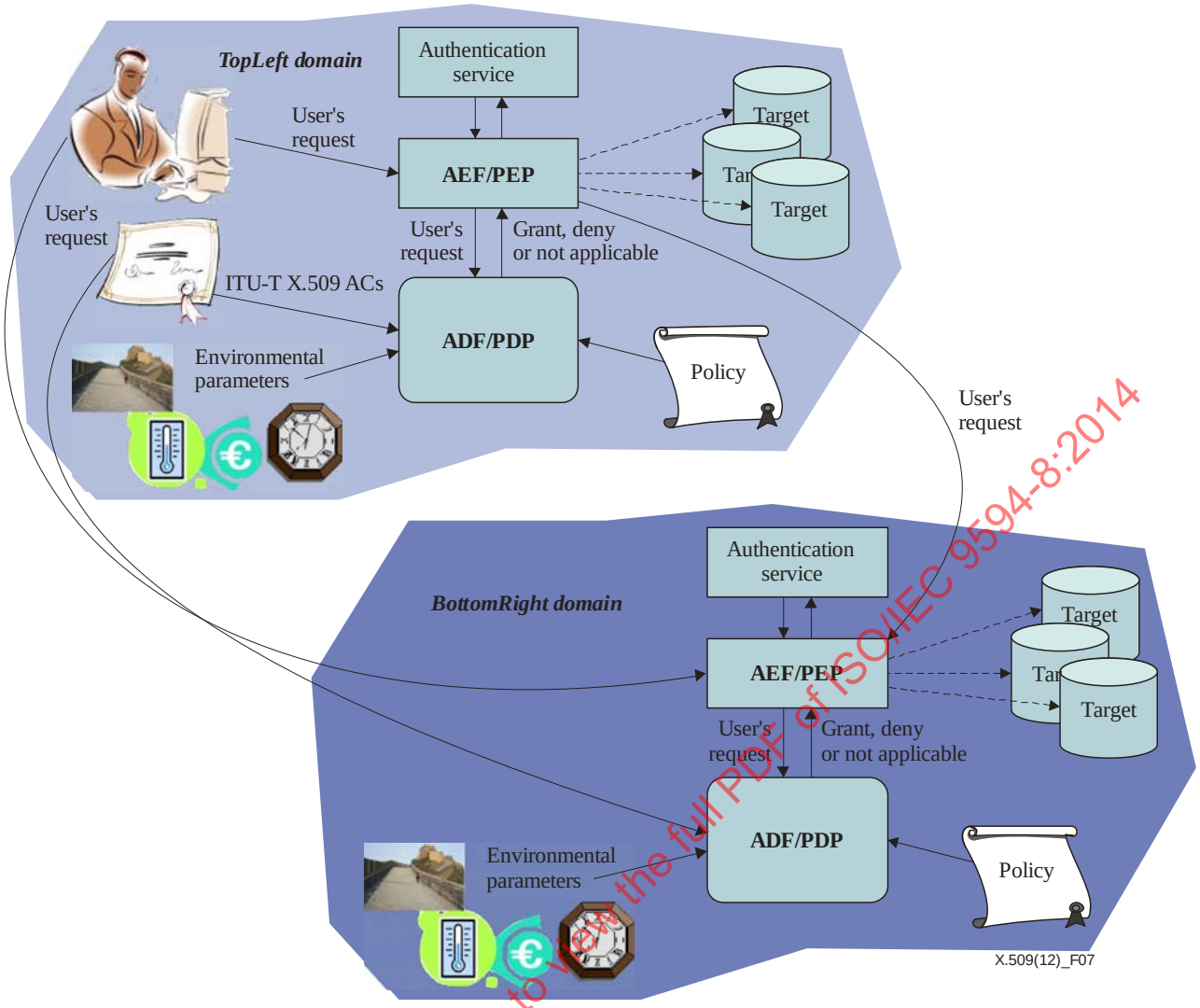


Figure 7 – Two federated PMI domains

In Figure 7, 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 ACs 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.

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 AC to the remote SOA or by the local SOA issuing an administrative role AC to the remote SOA so that the remote SOA may issue its own policy AC. 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 AC to the remote SOA, it may be as follows:

– the holder field identifies the SOA of the remote domain;

– the issuer field identifies the local SOA;

– the attributes of the AC 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 ACs 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 AC 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 AC in which the holder is the administrative role and the attribute is the **permission** attribute (defined in clause 14.8.1 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 AC 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 AC 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 AC.
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 AC, either a delegated role specification AC or a delegated attribute mapping AC. In a delegated role specification AC, 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 AC, 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 AC lie within the administrative scope defined for the administrative role. If they do, the delegated policy AC is accepted, and its policy rules become dynamically incorporated into the local SOA's policy. If they do not, the delegated policy AC is rejected, and its policy rules will be ignored.

A delegated role specification AC comprises:

- the holder is the newly specified remote role;
- the issuer field identifies the remote SOA (or AA) of the remote domain that issued this AC;
- the attributes of the AC 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 AC comprises:

- the holder and the issuer field identify the remote SOA (or AA) of the remote domain that issued this AC;
- the attributes field 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 ACs 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 AC issued to the remote SOA. The privilege attributes in the ACs 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 AC 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 AC to the remote SOA an attribute mapping extension can be placed in the policy AC 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 AC or place the attribute mapping extension in the ACs that it issues to its users.

If the remote SOA issues an attribute mapping policy AC, this should contain:

– a holder and issuer name which is that of the remote SOA;

– the attributes field is null;

– **attributeMappings** extension set to describe the attribute mappings.

NOTE – A remote SOA should not issue an attribute mapping AC 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 AC needs to be made available to the local PDPs at decision time. This can be done by either storing the policy AC 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 AC in the set of ACs presented by the remote user when accessing the local resource (the push model).

14.7 XML privilege information attribute

The specification of privileges is generally an application-specific issue that is outside the scope of this Directory 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>
```

14.8 Permission attribute and matching rule

14.8.1 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 Directory 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,
  ... }
```

The permission attribute is intended to be used to populate the **attributes** field of an attribute certificate and is not intended for storing as an attribute of a directory entry.

14.8.2 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.

15 Privilege management certificate 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 (ACRLs and AARLs) may contain any CRL or CRL entry extensions as defined for use in CRLs and CARLs in Section 2 of this Directory Specification.

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: 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.

15.1 Basic privilege management extensions

15.1.1 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 AC 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.

15.1.2 Basic privilege management extension fields

The following extension fields are defined:

a) Time specification;

b) Targeting information;

c) User notice;

d) Acceptable privilege policies;

e) Indirect issuer;

f) Single use;

g) Group AC.

15.1.2.1 Time specification extension

15.1.2.1.1 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 field is defined as follows:

```
timeSpecification EXTENSION ::= {
  SYNTAX         TimeSpecification
  IDENTIFIED BY  id-ce-timeSpecification }
```

This extension may be present in attribute certificates or public-key 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 certificates that contain the SOA identifier extension or in certificates issued to 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 time period during which the AA is able to issue certificates.

Because this extension is effectively specifying a refinement on the validity period of the certificate that contains it, this extension shall be marked 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 rejected.

15.1.2.1.2 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 stored value contains the **timeSpecification** extension and if components that are present in the presented value match the corresponding components of the stored value.

15.1.2.2 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 field is defined as follows.

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

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 **targetName** component, if present, provides the name of target servers/services for which the containing attribute certificate is targeted.

The **targetGroup** component, if present, provides 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 Directory Specification.

The **targetCert** component, if present, identifies 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 certificate. It does not impact the AA ability to issue certificates.

This extension is always critical.

If this extension is present, but the privilege verifier is not among those specified, the attribute certificate should be rejected.

If this extension is not present, then the attribute certificate is not targeted and may be accepted by any server.

15.1.2.3 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 field 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 public-key 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 certificates that contain the SOA identifier extension or in certificates issued to 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 certificates.

This extension may, at the option of the certificate issuer, be either critical or non-critical.

If this extension is flagged 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 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.

15.1.2.4 Acceptable privilege policies extension

The acceptable privilege policies field is used to constrain the assertion of the assigned privileges for use with a specific set of privilege policies.

This field 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 public-key certificates issued by AAs, including SOAs, to other AAs or to end-entities. 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 critical.

If this extension is present and the privilege verifier understands it, the 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 rejected.

15.1.2.5 Single use extension

In some scenarios, an AA may wish to issue an AC that can only be asserted once to a relying party within the lifetime of the AC. The **singleUse** extension is defined as follows:

```
singleUse EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY   id-ce-singleUse }
```

This extension may be present in ACs issued by AAs and SOAs to end-entities. This extension shall not be included in public-key certificates or in attribute certificates issued to AAs.

This extension is always critical.

Any relying party that accepts a **singleUse** AC should keep a record of at least the issuer and serial number of the AC, until after the expiry date of the AC in order to ensure that the holder cannot use the AC again. Ideally, all relying parties for which the AC is valid should have a coordination capability to ensure that the holder is not able to use the **singleUse** certificate with multiple relying parties. Alternatively the issuer of the **singleUse** AC should include a **targetingInformation** extension in the AC to limit the relying parties at which the AC is valid.

15.1.2.6 Group AC extension

In some scenarios it might be required for an AA to issue an AC 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 AC may be flagged as such by adding the group AC extension into the AC.

```
groupAC EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY   id-ce-groupAC }
```

This extension may or may not be critical. This extension shall only be added to end-entity ACs, and not to AA ACs or PKCs.

15.2 Privilege revocation extensions

15.2.1 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.

15.2.2 Privilege revocation extension fields

The following extension fields are defined:

a) CRL distribution points;

b) No revocation information.

15.2.2.1 CRL distribution points extension

The CRL distribution points extension is defined in Section 2 of this Directory Specification, for use in public-key certificates. This field may also be included in an attribute certificate. It may be present in certificates issued to AAs, including SOAs, as well as certificates issued to end-entities.

If present in a certificate, a privilege verifier shall process this extension in exactly the same manner as described in Section 2 for public-key certificates.

15.2.2.2 No revocation information extension

In some environments (e.g., where attribute certificates are issued with very short validity periods), there may not be a need to revoke certificates. An AA may use this extension to indicate that revocation status information is not provided for this attribute certificate. This field 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 end-entities. This extension shall not be included in public-key certificates or in attribute certificates issued to AAs.

This extension is always non-critical.

If this extension is present in an attribute certificate, a privilege verifier need not seek revocation status information.

15.3 Source of Authority extensions

15.3.1 Requirements

The following requirements relate to Sources of Authority:

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.

15.3.2 SOA extension fields

The following extension fields are defined:

a) SOA identifier;

b) Attribute descriptor.

15.3.2.1 SOA identifier extension

15.3.2.1.1 SOA identifier extension definition

The SOA identifier extension indicates that the certificate subject may act as an SOA for the purposes of privilege management. As such, the 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 certificates that assign privilege to holders. Those subsequent certificates may be attribute certificates or 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 field 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.

This field 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 or to end-entity privilege holders.

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 is always non-critical.

15.3.2.1.2 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 value contains an SOA Identifier extension.

15.3.2.2 Attribute descriptor extension

15.3.2.2.1 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 Directory 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** field;

– is a self-issued certificate (i.e., the issuer and holder are the same entity); and

– includes the attribute descriptor extension.

This field 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 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 the Directory, attribute certificates that contain this extension would be stored in the **attributeDescriptorCertificate** attribute of the SOA's directory entry.

15.3.2.2.2 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.

15.4 Role extensions

15.4.1 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.

15.4.2 Role extension fields

The following extension field is defined:

– Role specification certificate identifier.

15.4.2.1 Role specification certificate identifier extension

15.4.2.1.1 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 field may be used to locate that certificate.

This field 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 is always non-critical.

15.4.2.1.2 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.

15.5 Delegation extensions

15.5.1 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.

15.5.2 Delegation extension fields

The following extension fields 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;

g) No assertion.

15.5.2.1 Basic attribute constraints extension

15.5.2.1.1 Basic attribute constraints extension definition

This field 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 field 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 subject of this certificate may issue certificates only to end-entities and not to AAs. If no **pathLenConstraint** field 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 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 critical or 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 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 non-critical, and is not recognized by the privilege verifier, then that system should use other means to determine if the delegated attribute 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.

15.5.2.1.2 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.

15.5.2.2 Delegated name constraints extension

15.5.2.2.1 Delegated name constraints extension definition

The delegated name constraints field indicates a name space within which all holder names in subsequent certificates in a delegation path need to be located.

This field 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 critical or non-critical. It is recommended that it be flagged 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.

15.5.2.2.2 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.

15.5.2.3 Acceptable certificate policies extension

15.5.2.3.1 Acceptable certificate policies extension definition

The acceptable certificate policies field 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 field, 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 field 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 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 certificate shall be rejected.

15.5.2.3.2 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.

15.5.2.4 Authority attribute identifier extension

15.5.2.4.1 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 field 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 is always non-critical.

15.5.2.4.2 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.

15.5.2.5 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 field 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 is always non-critical.

The presence of this extension within an attribute certificate may be determined by applying the **extensionPresenceMatch** matching rule.

15.5.2.6 Issued on behalf of extension

This extension is inserted into an AC by an indirect issuer (DS server). It indicates the AA that has requested the DS server to issue the AC, 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 AC.

The issuer of this AC must have been granted the privilege to issue ACs on behalf of other AAs by an SOA, through the **IndirectIssuer** extension in its AC.

This extension may be critical or non-critical as necessary to ensure delegation path validation.

15.5.2.7 No assertion extension

If present, this extension indicates that the AC holder cannot assert the privileges indicated in the attributes of the AC. This field can only be inserted into AA ACs, and not into end-entity ACs. If present, this extension shall always be marked as being critical.

```
noAssertion EXTENSION ::= {
  SYNTAX          NULL
  IDENTIFIED BY   id-ce-noAssertion }
```

15.6 Recognition of Authority Extensions

15.6.1 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 users;

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.

15.6.2 RoA extension fields

The following extension fields are defined:

a) Allowed attribute assignments;

b) Attribute mappings;

c) Holder name constraints.

15.6.2.1 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-allowedAttAss }

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 being critical.

15.6.2.2 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.

15.6.2.3 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 ACs.

This extension indicates that constraints are being placed on the name forms and name spaces of all name forms in ACs issued by this AA and all subsequent AAs in the AC chain. If this extension is absent from all ACs in an AC chain, then no constraints are placed on any name spaces in the AC chain. If this extension is present in an AC certificate, then constraints are automatically placed on the name spaces of every name form in the AC 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 field 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 DN 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 ACs in a certification path shall be located in the permitted name spaces for the certificate 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 fields.

The **directoryName** name form satisfies this requirement; when using this name form, a naming subtree corresponds to a DIT subtree. An AC is considered subordinate to the **base** (and therefore a candidate to be within the subtree) if the sequence of RDNs, which forms the full DN in **base**, matches the initial sequence of the same number of RDNs which forms the first part of the DN of the holder of the AC. The DN of the holder of the certificate may have additional

trailing RDNs in its sequence that do not appear in the DN in **base**. The **distinguishedNameMatch** matching rule is used to compare the value of **base** with the initial sequence of RDNs in the DN of the subject of the certificate.

Conformant implementations are not required to recognize all possible name forms. If an AC using implementation does not recognize a name form used in any **base** component, and

- that name form also occurs in the **holder** field of a subsequent AC in the chain, then that AC shall be handled as if an unrecognized critical extension had been encountered; or
- that name form does not occur in the **holder** field of a subsequent AC in the chain, then this name form can be ignored.

If an AC using implementation does not recognize a name form that occurs in the **holder** field of a subsequent AC 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 AC shall be rejected.

This extension shall always be critical.

An AC using system shall check that the attribute certification path being processed is within the constraints specified by the value in this extension.

15.6.2.4 Relationship of delegated name constraints to holder name constraints

The **delegatedNameConstraints** extension described in clause 15.5.2.2 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 AC, 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.

16 Privilege path processing procedure

Privilege 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 privilege 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 16.1 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 16.2 below. If the privilege asserter was delegated its privilege by an intermediary AA, then the delegation path procedure in clause 16.3 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.

16.1 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 10. 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 AC issuers need have PKCs 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 Directory 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 AC is used to assert a privilege, the privilege verifier shall check that the AC has not been asserted prior to the current use.

If the certificate is a role assignment certificate, the processing procedure described in clause 16.2 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 16.3 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 16.2 and the delegation procedure in clause 16.3 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.

16.2 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 16.1 to determine whether or not the asserted privileges are sufficient for the context of use.

16.3 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 AC. 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 the 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 relying party should fully validate the delegation chain as follows:

i) Starting with the end entity AC, the RP extracts the issuer name and the **issuedOnBehalfOf** name.

ii) The RP retrieves the AC of the issuer and validates that the issuer is an indirect issuer of the SOA (i.e., has the **indirectIssuer** extension).

iii) The RP retrieves the AC 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 16.3.1 below.

The RP recurses to step ii) using the AC of the AA, and thereby moves up the chain until it arrives at the AC of an AA that is issued by the SOA.

16.3.1 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.

16.3.2 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 16.3.2.1 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 16.3.2.2 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 10.

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.

16.3.2.1 Use of authority information access extension

The authority information access (AIA) extension is defined in 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 AC 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 AC in which this extension appears. The type and format of the additional information is specified by the **AccessMethod** field; the **accessLocation** field 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 AC containing this extension. The referenced AC(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**, the **accessLocation** field describes the referenced description server and the access protocol to obtain the referenced ACs. The **accessLocation** field 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 AC containing this extension, but rather the **accessLocation** points to all the ACs belonging to the issuer of the AC containing this extension.

The ftp and http URIs should specify either the single DER encoded attribute certificate that was used to issue the AC containing this extension, or a filestore directory containing the set of ACs belonging to the issuer of the AC 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.

16.3.2.2 Use of authority key identifier

The AKI is used to identify the public key to be used to verify the signature on the AC 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 AC 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 16.3.2.1 above, 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 AC 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 AC issuer.

16.3.3 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 16.2. The comparison of the two privileges is done based on the domination rule discussed in clause 16.3.1.

16.3.4 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 16.1 to determine whether or not the direct privilege asserter has sufficient privilege for the context of use.

17 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.

17.1 PMI directory object classes

This subclause defines object class definitions for representing PMI objects in the Directory.

17.1.1 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 }
```

17.1.2 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 |
                 attributeAuthorityRevocationList}
  ID            id-oc-pmiAA }
```

17.1.3 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 |
                 attributeAuthorityRevocationList |
                 attributeDescriptorCertificate}
  ID            id-oc-pmiSOA }
```

17.1.4 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 |
                attributeAuthorityRevocationList}
  ID           id-oc-attCertCRLDistributionPts }
```

17.1.5 PMI delegation path

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 }
```

17.1.6 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 }
```

17.1.7 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 }
```

17.2 PMI Directory attributes

This subclause defines directory attributes used to store PMI data in directory entries.

17.2.1 Attribute certificate attribute

The following attribute contains 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 }
```

17.2.2 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 }
```

17.2.3 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 }
```

17.2.4 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
  ID                       id-at-attributeCertificateRevocationList }
```

17.2.5 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
  ID                       id-at-attributeAuthorityRevocationList }
```

17.2.6 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.

17.2.7 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.

17.2.8 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.

17.2.9 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 }
```

17.3 PMI general directory matching rules

This subclause defines matching rules for PMI directory attributes.

17.3.1 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.

17.3.2 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.

17.3.3 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.

17.3.4 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 field 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 field of the last certificate in the **SEQUENCE** in the stored value. This matching rule returns FALSE if either match fails.

17.3.5 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 – DIRECTORY USE OF PUBLIC-KEY & ATTRIBUTE CERTIFICATE FRAMEWORKS

The Directory uses the public-key certificate framework as the foundation for a number of security services including strong authentication and protection of Directory operations, as well as protection of stored data. The Directory uses the attribute certificate framework as the foundation for rule-based access control scheme. The relationship of the elements of the public-key certificate framework and of the attribute certificate framework to the various Directory security services is defined here. The specific security services provided by the Directory are fully specified over the complete set of Directory Specifications.

18 Directory authentication

The Directory supports the authentication of users accessing the Directory via DUAs and the authentication of directory systems (DSAs) to users and to other DSAs. Depending on the environment, either simple or strong authentication may be used. The procedures to be used for simple and strong authentication in the Directory are described in the following subclauses.

18.1 Simple authentication procedure

Simple authentication is intended to provide local authorization based upon the distinguished name of a user, a bilaterally agreed (optional) password, and a bilateral understanding of the means of using and handling this password within a single domain. The utilization of simple authentication is primarily intended for local use only, i.e., for peer entity authentication between one DUA and one DSA or between one DSA and one DSA. Simple authentication may be achieved by several means:

a) the transfer of the user's distinguished name and (optional) password in the clear (non-protected) to the recipient for evaluation;

b) the transfer of the user's distinguished name, password, and a random number and/or a timestamp, all of which are protected by applying a one-way function;

c) the transfer of the protected information described in b) together with a random number and/or a timestamp, all of which is protected by applying a one-way function.

NOTE 1 – There is no requirement that the one-way functions applied be different.

NOTE 2 – The signalling of procedures for protecting passwords may be a matter for an extension to the document.

Where passwords are not protected, a minimal degree of security is provided for preventing unauthorized access. It should not be considered a basis for secure services. Protecting the user's distinguished name and password provides greater degrees of security. The algorithms to be used for the protection mechanism are typically non-enciphering one-way functions that are very simple to implement.

The general procedure for achieving simple authentication is shown in Figure 8.

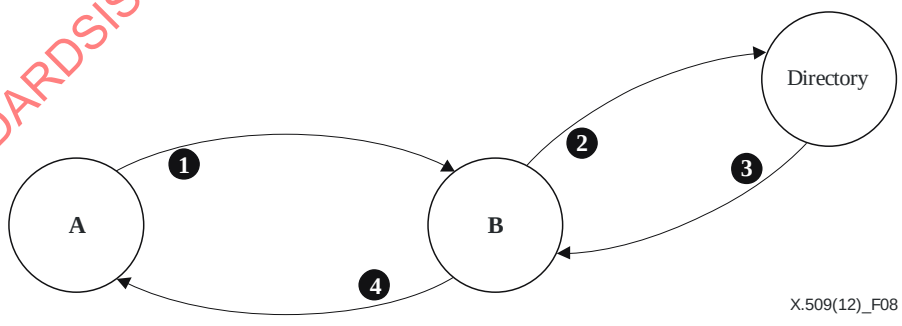


Figure 8 – The unprotected simple authentication procedure

The following steps are involved:

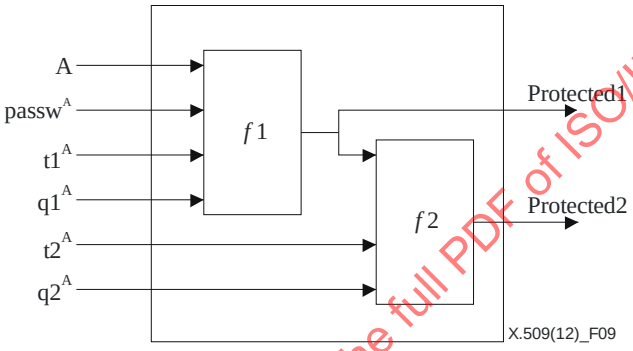
1) An originating user A sends its distinguished name and password to a recipient user B.
2) B sends the purported distinguished name and password of A to the Directory, where the password is checked against that held as the **`UserPassword`** attribute within the directory entry for A (using the Compare operation of the Directory).
3) The Directory confirms (or denies) to B that the credentials are valid.
4) The success (or failure) of authentication may be conveyed to A.

The most basic form of simple authentication involves only step 1) and after B has checked the distinguished name and password, may include step 4).

18.1.1 Generation of protected identifying information

Figure 9 illustrates two approaches by which protected identifying information may be generated. *f*1 and *f*2 are one-way functions (either identical or different) and the timestamps and random numbers are optional and subject to bilateral agreements.

Annex K provides a suggested algorithm to be used for protected passwords.



A	User's distinguished name
t^A	Timestamps
$passw^A$	Password of A
q^A	Random numbers, optionally with a counter included

Figure 9 – Protected simple authentication

18.1.2 Procedure for protected simple authentication

Figure 10 illustrates the procedure for protected simple authentication.

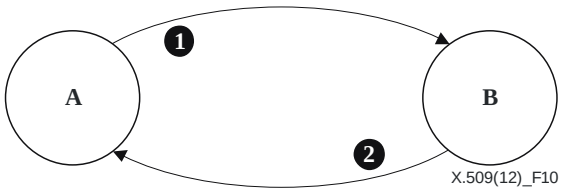


Figure 10 – The protected simple authentication procedure

The following steps are involved (initially using *f*1 only):

1) An originating user, user A, sends its protected identifying information (Authenticator1) to user B. Protection is achieved by applying the one-way function (f1) of Figure 9, where the timestamp and/or random number (when used) is used to minimize replay and to conceal the password.

   The protection of A's password is of the form:

   Protected1 = *f*1 ($t1^A$, $q1^A$, A, $passw^A$)

   The information conveyed to B is of the form:

   Authenticator1 = $t1^A$, $q1^A$, A, Protected1

2) B verifies the protected identifying information offered by A by generating (using the distinguished name and optional timestamp and/or random number provided by A, together with a local copy of A's password) a local protected copy of A's password (of the form Protected1). B compares for equality the purported identifying information (Protected1) with the locally generated value.

3) B confirms or denies to A the verification of the protected identifying information.

The procedure can be modified to afford greater protection using *f*1 and *f*2. The main differences are as follows:

1) A sends its additionally protected identifying information (Authenticator2) to B. Additional protection is achieved by applying a further one-way function, *f*2, as illustrated in Figure 9. The further protection is of the form:

Protected2 = $f2\ (t2^A, q2^A, \text{Protected1})$

The information conveyed to B is of the form:

Authenticator2 = $t1^A, t2^A, q1^A, q2^A$, A, Protected2

For comparison, B generates a local value of A's additionally protected password and compares it for equality with that of Protected2.

2) B confirms or denies to A the verification of the protected identifying information.

NOTE – The procedures defined in these clauses are specified in terms of A and B. As applied to the Directory (specified in Rec. ITU-T X.511 | ISO/IEC 9594-3 and Rec. ITU-T X.518 | ISO/IEC 9594-4), A could be a DUA binding to a DSA, B; alternatively, A could be a DSA binding to another DSA, B.

18.1.3 User Password attribute type

The multi-valued User Passwords attribute type contains the current and possibly previous passwords of an object. An attribute value for a user password is a string specified by the object.

```
userPassword ATTRIBUTE ::= {
  WITH SYNTAX               OCTET STRING(SIZE (0..MAX))
  EQUALITY MATCHING RULE    octetStringMatch
  LDAP-SYNTAX               octetString.&id
  LDAP-NAME                 {"userPassword"}
  ID                        id-at-userPassword }
```

18.2 Password policy

18.2.1 Introduction

Password policy is a set of rules that controls how passwords are used and administered in the Directory. It improves the security of the Directory and makes it difficult for password cracking programs to break into the Directory. These rules ensure that users change their passwords periodically, that passwords meet quality requirements, that the reuse of old password is restricted, and that users are locked out after a certain number of failed attempts. This policy also forces the user to update its password after it has been set for the first time, or has been reset by a password administrator. However, in some cases, it is desirable to disallow users from adding and updating their own passwords.

A password is supposed not to be well known. If a password is frequently changed, the chance of misuse is minimized. Password policy administrators may deploy a password policy that causes passwords to expire after a given amount of time thus forcing users to change their passwords periodically. There must be a way to make users aware of the need to change their password before being locked out of their accounts. One or both of the following methods could be used:

– A warning may be returned to the user sometime before the password is due to expire. If the user ignores this warning before the expiration time, the account will be locked.

– The user may Bind to the directory a certain number of times after the password has expired. If the user fails to change the password following one of the 'grace' authentications, the account will be locked.

Password quality rules are rules for how a password shall be constructed. It is not the intention to provide a specification for password qualities, as requirements on quality may change over time. Password quality includes things like:

– minimum length;

– a mixture of characters (uppercase, lowercase, figures, punctuations, etc.); and

– avoidance of trivial passwords.

A particular quality rule requires specialized code within the implementation. It may therefore be advantage to standardize password quality rules and assign object identifiers to such rules. An implementation may then claim support to one or more of such standardized quality rules.

An intruder may try to guess a password to get access to protected information. Currently, two different safeguards have been identified:

- Specification of the maximum number of failed attempts before a successful attempt within a given time span (which could be indefinitely). This approach allows for "denial of service attacks". One or more genuine users could have their access to the directory barred by the action of an attacker.
- The other mechanism is to insert a delay before returning information on authentication failure, and increasing this delay for repeated failed authentications on the same connection. This approach slows authentication, and makes brute force attacks impractical.

Password history is a mechanism to prevent password reuse. Previously used passwords should be stored to allow the Directory to ensure that a new password has not been previously used. Old passwords are stored for a time specified by the password policy, and after this time a password may be reused. The history is maintained in a **userPwdHistory** multi-valued operational attribute. A value is purged after a specific time, and the purged password may in principle be reused. The maximum time a password is kept in the in **userPwdHistory** attribute is specified in the pwdMaxTimeInHistory operational attribute, and the minimum time is specified in the pwdMinTimeInHistory operational attribute. The number of passwords stored is limited by the **pwdHistorySlots** operational attribute and the password cannot be changed if there is no free slot in the history and no passwords in the history have been for less than the pwdMinTimeInHistory, so a user cannot revert to a "preferred password" simply by making lots of password changes.

The password policy can be used with clear passwords (using the **clear** alternative of the **userPwd** attribute), or with encrypted passwords (using the **encrypted** alternative of the **userPwd** attribute) or with another password attribute. All entries in the same specific password administrative area shall use the same password attribute type.

18.2.2 Operational attributes and procedures

The password policy uses specific operational attributes to register policy parameters, times and dates related to password management.

When a password value is first stored in the directory in the **userPwd** attribute, the **pwdStartTime** operational attribute is set (Figure 11). The **pwdExpiryTime** operational attribute which contains the expiration of the password may either be automatically computed from the **pwdExpiryAge** operational attribute or set by explicit administrator action. It is an implementation option whether the value is dynamically computed by addition of the **pwdExpiryAge** to the **pwdStartTime** of the entry, in which case it does not need to be stored in the directory entry, or is set by an administrator, in which case it shall be stored in the directory entry. The **pwdEndTime** operational attribute which contains the expiration of the account may either be automatically computed from the **pwdMaxAge** operational attribute or set by explicit administrator action. It is an implementation option whether the value is dynamically computed by addition of the **pwdMaxAge** to the **pwdStartTime** of the entry, in which case it does not need to be stored in the directory entry, or is set by an administrator, in which case it shall be stored in the directory entry.

The **pwdStartTime** operational attribute may also be set by an Administrator to specify that the account cannot be used before a given time.

When the user (or an administrator acting on behalf of the user) changes the **userPwd** attribute within the **pwdMaxAge** period, the **pwdStartTime** operational attribute should be updated. The **pwdExpiryTime** and the **pwdEndTime** operational attributes should be recomputed and updated to reflect the new password creation time.

NOTE – If a user does bind with the Directory for a long time, the values of pwdExpiryTime and pwdEndTime operational attributes may be exceeded and the account automatically locked.

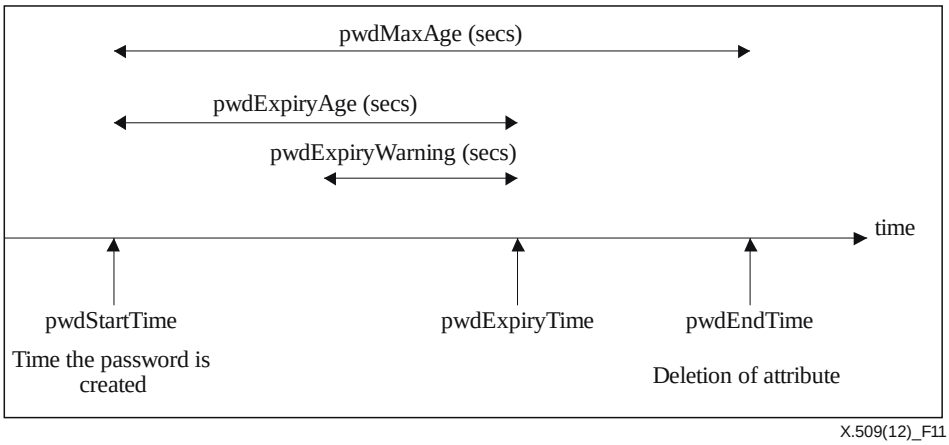


Figure 11 – Time chart for password attributes

When the user (or an administrator acting on behalf of the user) changes the value of the password, the new value is generally not known by all the Directory servers immediately because of replication delays. To prevent authentication problems, the previous password remains available for the **pwdRecentlyExpiredDuration** duration time (which shall be greater than the replication periods used in the Directory system).

When the user (or an administrator acting on behalf of the user) changes the value of the password, the old value should be copied into the recently expired password attribute. (The **userPwd** attribute is copied into the **userPwdRecentlyExpired**). When the recently expired password duration time is over, the recently expired password attribute (**userPwdRecentlyExpired**) should not be available to the user. If the user (or an administrator acting on behalf of the user) changes their password again during the recently expired password duration time, then their recently expired password should be overwritten and the duration should be set to start again (see Figure 12). Thus, a recently expired password will only be kept in the recently expired password attribute for the shorter of the recently expired password duration time or until the user changes their password again. However, it will be kept in the password history table.

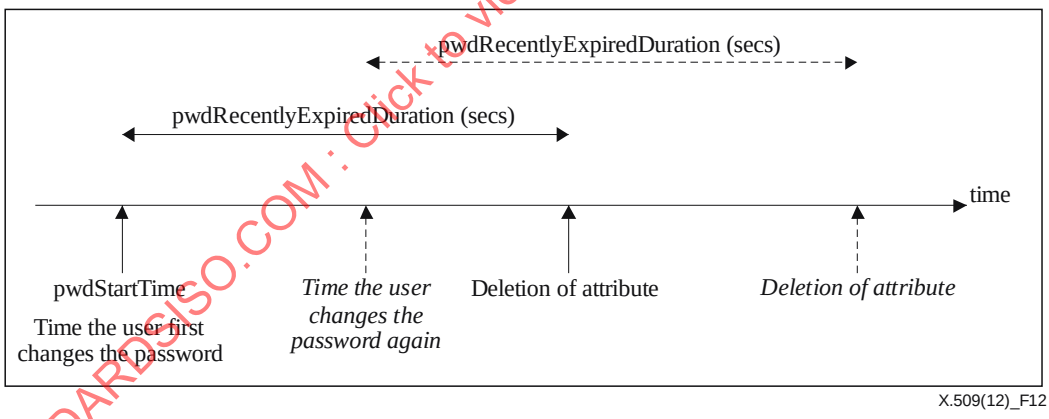


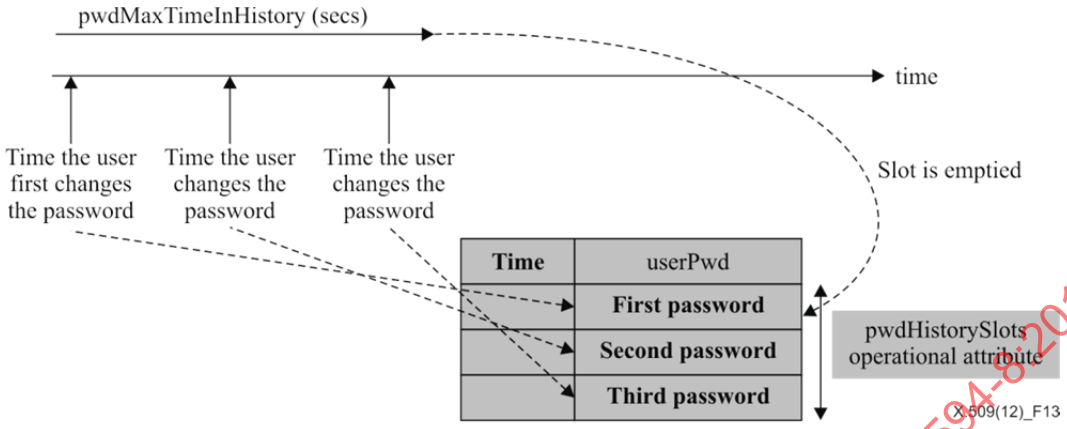
Figure 12 – Time line for recently expired passwords

18.2.3 Password history

The password history attribute is used to prevent password reuse, by storing old values of the user's password so that the user cannot reuse the same password again whilst it is stored in the password history (see Figure 13). When the user (or an administrator acting on behalf of the user) changes their password, it may be copied into the password history (**userPwdHistory**) operational attribute along with the time that the password was changed. The password maximum time in history attribute (pwdMaxTimeInHistory) specifies the maximum duration (in seconds) that a password should remain in the password history. Once this time has expired for a particular password, then it is removed from the password history, and the user may use this password again.

The number of slots in the password history table (or password history attribute values) is defined in the **pwdHistorySlots** operational attribute. When all the slots are filled, the oldest password may be removed subject to it having been in the history for a minimum duration time (as specified in the **pwdMinTimeInHistory** attribute).

NOTE – If an administrator has to change the password of a user when all the history slots are full and no password are older than **pwdMinTimeInHistory**, then the administrator might free two slots in the history table (i.e., delete two attribute values), reset the user's password to a temporary value (which is copied into the history), leaving one spare slot for the user to choose their own new password.



NOTE – If the Directory system changes its encryption algorithm then a user will be able to use the same password again since the encrypted password will be different.

Figure 13 – userPwdHistory attribute

18.2.4 Simple Authentication attributes held by object entries

18.2.4.1 User Password attribute

The **userPwd** attribute type contains either the clear text password or the encrypted password of an object. The Directory can store either variants but, implementations be aware that storing encrypted passwords is not always compatible with passing encrypted passwords in the protocol. The encrypted alternative may be used for passing the password in the bind or compare operations but this can only be safely used when the passwords are stored in the clear (see section 18.2.6.1 **userPwdMatch** for more details). The attribute value of the encrypted alternative is an octet string containing the encrypted value, with the encryption algorithm identifier, as well as parameters such as seeds. During password rollover, the old password value may be copied into the **userPwdRecentlyExpired** attribute value.

```
userPwd    ATTRIBUTE ::= {
  WITH SYNTAX                UserPwd
  EQUALITY MATCHING RULE     userPwdMatch
  SINGLE VALUE               TRUE
  LDAP-SYNTAX                userPwdDescription.&id
  LDAP-NAME                  {"userPwd"}
  ID                         id-at-userPwd }

UserPwd ::= CHOICE {
  clear                 UTF8String,
  encrypted             SEQUENCE {
    algorithmIdentifier   AlgorithmIdentifier{{SupportedAlgorithms}},
    encryptedString       OCTET STRING,
    ...},
  ...}
```

Annex L contains examples of some encryption methods.

18.2.4.2 Password Start Time attribute

The **pwdStartTime** operational attribute indicates when the password has been created for the object represented by the entry in which the attribute is present.

```
pwdStartTime ATTRIBUTE ::= {
  WITH SYNTAX                GeneralizedTime
  EQUALITY MATCHING RULE     generalizedTimeMatch
  ORDERING MATCHING RULE     generalizedTimeOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                generalizedTime.&id
  LDAP-NAME                  {"pwdStartTime"}
  ID                         id-oa-pwdStartTime }
```

18.2.4.3 Password Expiry Time attribute

The **pwdExpiryTime** operational attribute indicates when the password will expire for the object represented by the entry in which the attribute is present. This is an optional attribute that can be set by an administrator. If the attribute is missing, its default value is computed by the addition of the **pwdExpiryAge** to the **pwdStartTime** of the entry.

```
pwdExpiryTime ATTRIBUTE ::= {
  WITH SYNTAX                GeneralizedTime
  EQUALITY MATCHING RULE     generalizedTimeMatch
  ORDERING MATCHING RULE     generalizedTimeOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                generalizedTime.&id
  LDAP-NAME                  {"pwdExpiryTime"}
  ID                         id-oa-pwdExpiryTime }
```

18.2.4.4 Password End Time attribute

The **pwdEndTime** operational attribute indicates when the password will be no longer valid for the object represented by the entry in which the attribute is present. This is an optional attribute that can be set by an administrator. If the attribute is missing, its default value is computed by the addition of the **pwdMaxAge** to the **pwdStartTime** of the entry.

```
pwdEndTime ATTRIBUTE ::= {
  WITH SYNTAX                GeneralizedTime
  EQUALITY MATCHING RULE     generalizedTimeMatch
  ORDERING MATCHING RULE     generalizedTimeOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                generalizedTime.&id
  LDAP-NAME                  {"pwdEndTime"}
  ID                         id-oa-pwdEndTime }
```

18.2.4.5 Password Fails attribute

The **pwdFails** operational attribute specifies the current number of consecutive failed bind or compare attempts on the password attribute. The value of this attribute is incremented by one after a failed bind or compare attempt and is reset to zero after a successful bind or compare operation.

```
pwdFails ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (0..MAX)
  EQUALITY MATCHING RULE     integerMatch
  ORDERING MATCHING RULE     integerOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      dSAOperation
  LDAP-SYNTAX                integer.&id
  LDAP-NAME                  {"pwdFails"}
  ID                         id-oa-pwdFails }
```

18.2.4.6 Password Failure Time attribute

The **pwdFailureTime** operational attribute specifies the time of the last failed bind or compare attempts on the password attribute. This attribute is only significant when the pwdFails operational attribute contains a non zero value.

```
pwdFailureTime ATTRIBUTE ::= {
  WITH SYNTAX                GeneralizedTime
  EQUALITY MATCHING RULE     generalizedTimeMatch
  ORDERING MATCHING RULE     generalizedTimeOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      dSAOperation
  LDAP-SYNTAX                generalizedTime.&id
  LDAP-NAME                  {"pwdFailureTime"}
  ID                         id-oa-pwdFailureTime }
```

18.2.4.7 Password Graces Used attribute

The **pwdGracesUsed** operational attribute specifies the number of grace authentication attempts that have already been used with an expired password. The value of this attribute is set to 0 when the password is changed and incremented by one after successful authentication using an expired password. When the value is greater or equal to the **pwdGraces** attribute, the password is not usable again.

```
pwdGracesUsed ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (0..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     dSAOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdGracesUsed"}
  ID                        id-oa-pwdGracesUsed }
```

18.2.4.8 User Password History attribute

The **userPwdHistory** operational attribute is used to hold previous passwords for the user represented by the entry in which the attribute is present.

```
userPwdHistory ATTRIBUTE ::=
                    pwdHistory{userPwd,userPwdHistoryMatch,id-oa-userPwdHistory}
```

This attribute is multi-valued. Each value consists of a sequence of the time the password was put in the history and the password.

18.2.4.9 User Password Recently Expired attribute

The **userPwdRecentlyExpired** attribute type contains the old user password after it has been replaced during the **pwdRecentlyExpiredDuration**. During this period, this password and the **userPwd** attribute are both considered to be valid. This attribute is removed when the **pwdRecentlyExpiredDuration** expires.

```
userPwdRecentlyExpired ATTRIBUTE ::=
                    pwdRecentlyExpired{userPwd,id-oa-userPwdRecentlyExpired}
```

18.2.5 Password policy attributes

Password policy attributes may be placed in an object entry and/or in a subentry. If an object entry holds such an attribute and is also within the scope of a password administration subentry, the value of the attribute in the object entry itself takes precedence.

18.2.5.1 Passwsord ModifyEntry Allowed attribute

The **pwdModifyEntryAllowed** operational attribute specifies if the password or the encrypted password of an entry can be modified by an Administrator with a Modify Entry operation. If this attribute is missing, or the value is FALSE, the password or the encrypted password cannot be modified with a Modify Entry operation.

```
pwdModifyEntryAllowed ATTRIBUTE ::= {
  WITH SYNTAX               BOOLEAN
  EQUALITY MATCHING RULE    booleanMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               boolean.&id
  LDAP-NAME                 {"pwdModifyEntryAllowed"}
  ID                        id-oa-pwdModifyEntryAllowed }
```

18.2.5.2 Password Change Allowed attribute

The **pwdChangeAllowed** operational attribute specifies if the password or the encrypted password of an entry can be modified by the owner of that entry with a Change Password operation. If this attribute is missing or the value is FALSE, the password or the encrypted password cannot be modified with a Change Password operation.

```
pwdChangeAllowed ATTRIBUTE ::= {
  WITH SYNTAX               BOOLEAN
  EQUALITY MATCHING RULE    booleanMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               boolean.&id
  LDAP-NAME                 {"pwdChangeAllowed"}
  ID                        id-oa-pwdChangeAllowed }
```

18.2.5.3 Password Maximum Age attribute

The **pwdMaxAge** operational attribute holds the number of seconds after which a password will be no longer available. It shall have a value greater than zero.

If this attribute is missing, then the default value is infinity.

```
pwdMaxAge ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (1 .. MAX)
  EQUALITY MATCHING RULE     integerMatch
  ORDERING MATCHING RULE     integerOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                integer.&id
  LDAP-NAME                  {"pwdMaxAge"}
  ID                         id-oa-pwdMaxAge }
```

18.2.5.4 Password Expiry Age attribute

The **pwdExpiryAge** operational attribute holds the number of seconds after which a modified password will expire. It shall have a value greater than zero.

If this attribute is missing, then the default value is infinity.

```
pwdExpiryAge ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (1 .. MAX)
  EQUALITY MATCHING RULE     integerMatch
  ORDERING MATCHING RULE     integerOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                integer.&id
  LDAP-NAME                  {"pwdExpiryAge"}
  ID                         id-oa-pwdExpiryAge }
```

18.2.5.5 Password Quality Rule attributes

18.2.5.5.1 Password Minimum Length attribute

This specifies the minimum length, in characters, which is acceptable for a password.

```
pwdMinLength ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (0..MAX)
  EQUALITY MATCHING RULE     integerMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                integer.&id
  LDAP-NAME                  {"pwdMinLength"}
  ID                         id-oa-pwdMinLength }
```

18.2.5.5.2 Password Vocabulary attribute

This specifies the type of words that are forbidden to be used for passwords. If a bit is set, the corresponding type of word is not allowed to be used on its own as a password.

```
pwdVocabulary ATTRIBUTE ::= {
  WITH SYNTAX                PwdVocabulary
  EQUALITY MATCHING RULE     bitStringMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                pwdVocabularyDescription.&id
  LDAP-NAME                  {"pwdVocabulary"}
  ID                         id-oa-pwdVocabulary }

PwdVocabulary ::=  BIT STRING {
    noDictionaryWords   (0),
    noPersonNames       (1),
    noGeographicalNames (2) }
```

18.2.5.5.3 Password Alphabet attribute

This specifies the sets of characters that shall be used in creating a password. The password shall contain at least one character of each UTF8String of the value.

```
pwdAlphabet ATTRIBUTE ::= {
  WITH SYNTAX                PwdAlphabet
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
```

```
  LDAP-SYNTAX                pwdAlphabetDescription.&id
  LDAP-NAME                  {"pwdAlphabet"}
  ID                         id-oa-pwdAlphabet }

PwdAlphabet ::= SEQUENCE OF UTF8String
```

18.2.5.5.4 Password Dictionaries attribute

This attributes points to one or more dictionaries containing words that are forbidden from being passwords on their own.

```
pwdDictionaries ATTRIBUTE ::= {
  SUBTYPE OF                 uri
  USAGE                      directoryOperation
  LDAP-SYNTAX                directoryString.&id
  LDAP-NAME                  {"pwdDictionaries"}
  ID                         id-oa-pwdDictionaries }
```

18.2.5.6 Password Expiry Warning attribute

The **pwdExpiryWarning** operational attribute specifies a period in seconds before password expiration. During this period a warning indication shall be returned whenever an authenticating requester binds. If this attribute is missing, then a warning indication shall not be returned.

```
pwdExpiryWarning ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (1..MAX)
  EQUALITY MATCHING RULE     integerMatch
  ORDERING MATCHING RULE     integerOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                integer.&id
  LDAP-NAME                  {"pwdExpiryWarning"}
  ID                         id-oa-pwdExpiryWarning }
```

If the user does not attempt to bind during this period, the account should be locked, but the user should have a chance to change the password.

18.2.5.7 Password Graces attribute

The **pwdGraces** operational attribute specifies the number of times an expired password can be used to authenticate. If this attribute is missing, authentication shall fail.

```
pwdGraces ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (0..MAX)
  EQUALITY MATCHING RULE     integerMatch
  ORDERING MATCHING RULE     integerOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                integer.&id
  LDAP-NAME                  {"pwdGraces"}
  ID                         id-oa-pwdGraces }
```

18.2.5.8 Password Failure Duration attribute

The **pwdFailureDuration** operational attribute holds the number of seconds a response to a failed bind or compare attempt should be delayed.

```
pwdFailureDuration ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (0..MAX)
  EQUALITY MATCHING RULE     integerMatch
  ORDERING MATCHING RULE     integerOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                integer.&id
  LDAP-NAME                  {"pwdFailureDuration"}
  ID                         id-oa-pwdFailureDuration }
```

18.2.5.9 Password Lockout Duration attribute

The **pwdLockoutDuration** operational attribute holds the number of seconds that the password cannot be used to authenticate due to too many successive failed bind or compare attempts (more than the limit specified by **pwdMaxFailures** operational attribute or its default). If this attribute is missing, the default time is infinity.

```
pwdLockoutDuration ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (0..MAX)
  EQUALITY MATCHING RULE     integerMatch
  ORDERING MATCHING RULE     integerOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                integer.&id
  LDAP-NAME                  {"pwdLockoutDuration"}
  ID                         id-oa-pwdLockoutDuration }
```

18.2.5.10 Password Maximum Failures attribute

The **pwdMaxFailures** operational attribute specifies the number of consecutive failed bind or compare attempts after which the password may not be used to authenticate. If this attribute is missing, there is no limit on failed attempts.

```
pwdMaxFailures ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (1..MAX)
  EQUALITY MATCHING RULE     integerMatch
  ORDERING MATCHING RULE     integerOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                integer.&id
  LDAP-NAME                  {"pwdMaxFailures"}
  ID                         id-oa-pwdMaxFailures }
```

18.2.5.11 Password Maximum Time in History attribute

The **pwdMaxTimeInHistory** operational attribute specifies the maximum time, in number of seconds, during which a replaced password is kept within the **userPwdHistory** operational attribute. If this attribute is missing, the default is infinity.

```
pwdMaxTimeInHistory ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (1..MAX)
  EQUALITY MATCHING RULE     integerMatch
  ORDERING MATCHING RULE     integerOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                integer.&id
  LDAP-NAME                  {"pwdMaxTimeInHistory"}
  ID                         id-oa-pwdMaxTimeInHistory }
```

18.2.5.12 Password Minimum Time in History attribute

The **pwdMinTimeInHistory** operational attribute specifies the minimum time, in number of seconds, during which a replaced password shall be kept within the **userPwdHistory** operational attribute. If this attribute is missing, the default time is zero seconds.

```
pwdMinTimeInHistory ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (0..MAX)
  EQUALITY MATCHING RULE     integerMatch
  ORDERING MATCHING RULE     integerOrderingMatch
  SINGLE VALUE               TRUE
  USAGE                      directoryOperation
  LDAP-SYNTAX                integer.&id
  LDAP-NAME                  {"pwdMinTimeInHistory"}
  ID                         id-oa-pwdMinTimeInHistory }
```

18.2.5.13 Password History Slots attribute

The **pwdHistorySlots** operational attribute specifies the number of slots in the history which can be used to store replaced passwords. The minimum number of slots is 2 because two slots are needed when an administrator has to reset a password.

```
pwdHistorySlots ATTRIBUTE ::= {
  WITH SYNTAX                INTEGER (2..MAX)
```

```
  EQUALITY MATCHING RULE   integerMatch
  ORDERING MATCHING RULE   integerOrderingMatch
  SINGLE VALUE             TRUE
  USAGE                    directoryOperation
  LDAP-SYNTAX              integer.&id
  LDAP-NAME                {"pwdHistorySlots"}
  ID                       id-oa-pwdHistorySlots }
```

18.2.5.14 Password Recently Expired Duration attribute

The **pwdRecentlyExpiredDuration** attribute type defines the period in seconds during which an expired password is kept in the **userPwdRecentlyExpired** attribute.

```
pwdRecentlyExpiredDuration ATTRIBUTE ::= {
  WITH SYNTAX              INTEGER (0..MAX)
  EQUALITY MATCHING RULE   integerMatch
  ORDERING MATCHING RULE   integerOrderingMatch
  SINGLE VALUE             TRUE
  USAGE                    directoryOperation
  LDAP-SYNTAX              integer.&id
  LDAP-NAME                {"pwdRecentlyExpiredDuration"}
  ID                       id-oa-pwdRecentlyExpiredDuration }
```

18.2.5.15 Password Encryption Algorithm attribute

The **pwdEncAlg** operational attribute indicates the algorithm to be used during the creation of an encrypted password.

```
pwdEncAlg ATTRIBUTE ::= {
  WITH SYNTAX              PwdEncAlg
  EQUALITY MATCHING RULE   pwdEncAlgMatch
  SINGLE VALUE             TRUE
  USAGE                    directoryOperation
  LDAP-SYNTAX              integer.&id
  LDAP-NAME                {"pwdEncAlg"}
  ID                       id-oa-pwdEncAlg }

PwdEncAlg ::= AlgorithmIdentifier{{SupportedAlgorithms}}
```

The algorithms specified in **pwdEncAlg** shall be defined in Annex L.

18.2.6 Password policy matching rules

18.2.6.1 User Password matching rule

The **userPwdMatch** rule determines whether a presented clear text or encrypted password matches a clear text password stored in the Directory.

```
userPwdMatch MATCHING-RULE ::= {
  SYNTAX       UserPwd
  LDAP-SYNTAX  userPwdDescription.&id
  LDAP-NAME    {"userPwdMatch"}
  ID           id-mr-userPwdMatch }
```

It the presented password is clear text and the stored password is clear text, then comparison is performed using **caseExactMatch.**

If the presented password is clear text and the stored password is encrypted, then the clear text assertion is encrypted using the algorithm identified in the stored password and the encrypted value is compared with the stored value using **octetStringMatch**.

If the presented password is encrypted and the stored password is clear text, then comparison is performed by encrypting the stored password using the encryption algorithm passed in the assertion and then the encrypted password is compared to the asserted encrypted password using **octetStringMatch**.

If the presented password is encrypted and the stored password is encrypted, then the algorithm identifier and algorithm parameters are compared for equality. If they are different, matching fails. If they are the same, the encrypted passwords are compared using **octetStringMatch**.

18.2.6.2 Password Encryption Algorithm matching rule

The **pwdEncAlgMatch** rule compares for equality a presented password encryption algorithm with the algorithm stored with an encrypted password. The encryption algorithms are equal only if the algorithm identifiers and algorithm parameters are equals.

```
pwdEncAlgMatch MATCHING-RULE ::= {
  SYNTAX        PwdEncAlg
  LDAP-SYNTAX   pwdEncAlgDescription.&id
  LDAP-NAME     {"pwdEncAlgMatch"}
  ID            id-mr-pwdEncAlgMatch }
```

18.2.6.3 User Password History matching rule

The **userPwdHistoryMatch** rule compares for equality a presented clear or encrypted password with a clear text or encrypted password stored as an attribute value of type **pwdHistory**. The timestamp component present in the **userPwdHistory** is ignored. The remaining passwords are compared using the **userPwdMatch** matching rule.

```
userPwdHistoryMatch MATCHING-RULE ::= pwdHistoryMatch{userPwd,id-mr-userPwdHistoryMatch}
```

18.3 Strong Authentication

Strong authentication makes use of PKI as specified by this Directory Specification, which provides the basic approach to authentication. However, many authentication procedures employing this approach are possible. In general, it is the business of a specific application to determine the appropriate procedures, so as to meet the security policy of the application. This clause describes three particular authentication procedures which may be found useful across a range of applications.

NOTE – This Directory Specification does not specify the procedures to the detail required for implementation. However, additional standards could be envisaged which would do so, either in an application-specific or in a general-purpose way.

The three procedures involve different numbers of exchanges of authentication information, and consequently provide different types of assurance to their participants. Specifically:

a) One-way authentication, described in clause 18.3.1, involves a single transfer of information from one user (A) intended for another (B), and establishes the following:
   - the identity of A, and that the authentication token actually was generated by A;
   - the identity of B, and that the authentication token actually was intended to be sent to B;
   - the integrity and "originality" (the property of not having been sent two or more times) of the authentication token being transferred.

   The latter properties can also be established for arbitrary additional data accompanying the transfer.

b) Two-way authentication, described in clause 18.3.2, involves, in addition, a reply from B to A. It establishes, in addition, the following:
   - that the authentication token generated in the reply actually was generated by B and was intended to be sent to A;
   - the integrity and originality of the authentication token sent in the reply;
   - (optionally) the mutual secrecy of part of the tokens.

c) Three-way authentication, described in clause 18.3.3, involves, in addition, a further transfer from A to B. It establishes the same properties as the two-way authentication, but does so without the need for association timestamp checking.

In each case where Strong Authentication is to take place, A shall obtain the public key of B, and the return certification path from B to A, prior to any exchange of information. This may involve access to the Directory, as described in clause 18.2. Any such access is not mentioned again in the description of the procedures below.

The checking of timestamps as mentioned in the following clauses only applies when either synchronized clocks are used in a local environment, or if clocks are logically synchronized by bilateral agreements. In any case, it is recommended that Coordinated Universal Time be used.

For each of the three authentication procedures described below, it is assumed that party A has checked the validity of all of the certificates in the certification path.

18.3.1 One-way authentication

The following steps are involved, as depicted in Figure 14:

1) A generates r^A, a non-repeating number, which is used to detect replay attacks and to prevent forgery.
2) A sends the following message to B:

$$BA, A\{t^A, r^A, B\}$$

where t^A is a timestamp. t^A consists of one or two dates: the generation time of the token (which is optional) and the expiry date. Alternatively, if data origin authentication of "sgnData" is to be provided by the digital signature:

$$BA, A\{t^A, r^A, B, sgnData\}$$

In cases where information is to be conveyed which will subsequently be used as a private key (this information is referred to as "encData"):

$$BA, A\{t^A, r^A, B, sgnData, Bp[encData]\}$$

The use of "encData" as a private key implies that it shall be chosen carefully, e.g., to be a strong key for whatever cryptosystem is used as indicated in the "sgnData" field of the token.

3) B carries out the following actions:
   a) obtains Ap from BA, checking that A's certificate has not expired;
   b) verifies the signature, and thus the integrity of the signed information;
   c) checks that B itself is the intended recipient;
   d) checks that the timestamp is "current";
   e) optionally, checks that r^A has not been replayed. This could, for example, be achieved by having r^A include a sequential part that is checked by a local implementation for its value uniqueness.

      r^A is valid until the expiry date indicated by t^A. r^A is always accompanied by a sequential part, which indicates that A shall not repeat the token during the timerange t^A and therefore that checking of the value of r^A itself is not required.

      In any case, it is reasonable for party B to store the sequential part together with timestamp t^A in the clear and together with the hashed part of the token during timerange t^A.

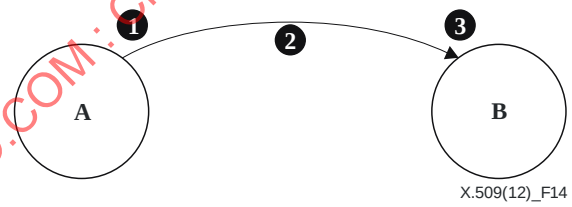


Figure 14 – One-way authentication

18.3.2 Two-way authentication

The following steps are involved, as depicted in Figure 15:

1) as for clause 18.3.1;
2) as for clause 18.3.1;
3) as for clause 18.3.1;
4) B generates r^B, a non-repeating number, used for similar purpose(s) to r^A;
5) B sends the following authentication token to A:

$$B\{t^B, r^B, A, r^A\}$$

where t^B is a timestamp defined in the same way as t^A.

Alternatively, if data origin authentication of "sgnData" is to be provided by the digital signature:

$$B\{t^B, r^B, A, r^A, sgnData\}$$

In cases where information is to be conveyed which will subsequently be used as a private key (this information is referred to as "encData"):

$$B\{t^B, r^B, A, r^A, sgnData, Ap[encData]\}$$

The use of "encData" as a private key implies that it shall be chosen carefully, e.g., to be a strong key for whatever cryptosystem is used as indicated in the "sgnData" field of the token.

6) A carries out the following actions:
   a) verifies the signature, and thus the integrity of the signed information;
   b) checks that A is the intended recipient;
   c) checks that the timestamp t^B is "current";
   d) optionally, checks that r^B has not been replayed (see clause 18.3.1, step 3), d)).

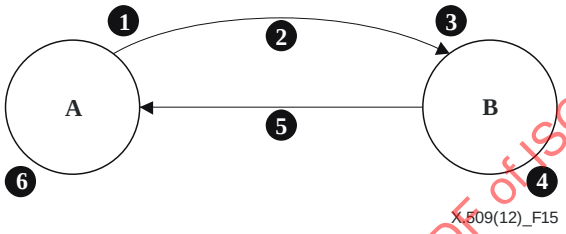


Figure 15 – Two-way authentication

18.3.3 Three-way authentication

The following steps are involved, as depicted in Figure 16:

1) as for clause 18.3.2;
2) as for clause 18.3.2. Timestamp t^A may be zero;
3) as for clause 18.3.2, except that the timestamp need not be checked;
4) as for clause 18.3.2;
5) as for clause 18.3.2. Timestamp t^B may be zero;
6) as for clause 18.3.2, except that the timestamp need not be checked;
7) A checks that the received r^A is identical to the r^A which was sent;
8) A sends the following authentication token to B:

$$A\{r^B, B\}$$

9) B carries out the following actions:
   a) checks the signature and thus, the integrity of the signed information;
   b) checks that the received r^B is identical to the r^B which was sent by B.

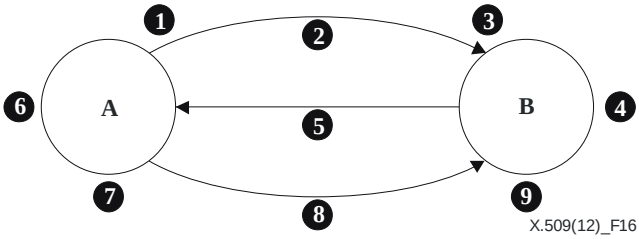


Figure 16 – Three-way authentication

19 Access control

The Directory exists in an environment where various administrative authorities control access to their portion of the DIB. The definition of an access control scheme in the context of the Directory includes methods to:

- specify access control information (ACI);
- enforce access rights defined by that access control information;
- maintain access control information.

The enforcement of access rights applies to controlling access to:

- Directory information related to names;
- Directory user information;
- Directory operational information including access control information.

Administrative authorities may make use of all or parts of any standardized access control scheme in implementing their security policies, or may freely define their own schemes at their discretion.

The Basic Access Control (BAC) scheme defined in Rec. ITU-T X.501 | ISO/IEC 9594-2 is an access control list based scheme that enables Directory Administrators to tie permissions to the level of authentication performed to bind to the Directory. The public-key certificate framework defined in this Directory Specification is used to provide the strong authentication scheme used for this binding.

The Rules Based Access Control (RBAC) scheme defined in Rec. ITU-T X.501 | ISO/IEC 9594-2 makes use of the attribute certificate framework defined in this Directory Specification to carry clearance attributes used in making access control decisions. RBAC may also be used in conjunction with BAC.

20 Protection of Directory operations

The public-key certificate framework defined in this Directory Specification is used in all Directory protocols defined in these Directory Specifications to optionally protect the operations including requests, responses and errors. Integrity protection is provided through the digital signature of the sender and the verification of that signature by the recipient using the sender's corresponding public-key certificate. Confidentiality may be provided through the use of public key encryption where the content is encrypted with the public key obtained from the intended recipient's public-key certificate and decrypted by the recipient using its corresponding private key.

NOTE - Encryption and decryption using asymmetric keys as indicated above is known to be less efficient than using symmetric keys. However, these Directory Specifications do not provide the means for end-to-end encryption using symmetric keys.

The specific mechanisms and syntax for requesting and including the protection elements in protocol exchanges are defined within each of the Directory protocols in these Directory Specifications.

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 Directory Specification in the form of four ASN.1 modules: **AuthenticationFramework**, **CertificateExtensions,** and **AttributeCertificateDefinitions** and **PasswordPolicy**.

```
-- A.1 - Authentication framework module

AuthenticationFramework {joint-iso-itu-t ds(5) module(1) authenticationFramework(7) 7}
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-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) 7}

  ATTRIBUTE, DistinguishedName, MATCHING-RULE, Name, NAME-FORM, OBJECT-CLASS,
  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

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 } )

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-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 -- } )

SIGNATURE{ToBeSigned} ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}},
```

```
  encrypted            ENCRYPTED-HASH{ToBeSigned},
  ... }

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

-- 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]]
}

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

CertificateSerialNumber ::= INTEGER

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

-- Definition of the following information object set is deferred, perhaps to
-- standardized profiles or to protocol implementation conformance statements. The
-- set is required to specify a table constraint on the parameters component of
-- AlgorithmIdentifier.

SupportedAlgorithms ALGORITHM ::= {...}

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

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

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

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

Extensions ::= SEQUENCE 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 }

-- 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 |
                         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 |
                         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 }

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}

-- 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}


END - AuthenticationFramework

-- A.2 - Certificate extensions module

CertificateExtensions {joint-iso-itu-t ds(5) module(1) certificateExtensions(26) 7}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS ALL

IMPORTS
  id-at, id-ce, id-ldx, id-mr, informationFramework, authenticationFramework,
  selectedAttributeTypes
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 7}

  Name, RelativeDistinguishedName, Attribute{}, MATCHING-RULE,
  SupportedAttributes, SYNTAX-NAME
    FROM InformationFramework informationFramework

  CertificateSerialNumber, CertificateList, AlgorithmIdentifier{}, EXTENSION,
  Time, PolicyID, SupportedAlgorithms
    FROM AuthenticationFramework authenticationFramework

  UnboundedDirectoryString
    FROM SelectedAttributeTypes selectedAttributeTypes

  ORAddress
    FROM MTSAbstractService {joint-iso-itu-t mhs(6) mts(3) modules(0)
      mts-abstract-service(1) version-1999(1)};

-- 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,
  ... }

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),
  ... }

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

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) }

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,
  ... }

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-subjectDirectoryAttributes  OBJECT IDENTIFIER ::= {id-ce 9}
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-instructionCode             OBJECT IDENTIFIER ::= {id-ce 23}
id-ce-invalidityDate              OBJECT IDENTIFIER ::= {id-ce 24}
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-cRLStreamIdentifier         OBJECT IDENTIFIER ::= {id-ce 40}
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-baseUpdateTime              OBJECT IDENTIFIER ::= {id-ce 51}
id-ce-deltaInfo                   OBJECT IDENTIFIER ::= {id-ce 53}
id-ce-inhibitAnyPolicy            OBJECT IDENTIFIER ::= {id-ce 54}
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-aAissuingDistributionPoint  OBJECT IDENTIFIER ::= {id-ce 63}

-- 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}

-- The following OBJECT IDENTIFIERS are not used by this Specification:
-- {id-ce 2}, {id-ce 3}, {id-ce 4}, {id-ce 5}, {id-ce 6}, {id-ce 7},
-- {id-ce 8}, {id-ce 10}, {id-ce 11}, {id-ce 12}, {id-ce 13},
-- {id-ce 22}, {id-ce 25}, {id-ce 26}

END -- CertificateExtensions

--  A.3 - Attribute Certificate Framework module

AttributeCertificateDefinitions {joint-iso-itu-t ds(5) module(1)
  attributeCertificateDefinitions(32) 7}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS ALL

IMPORTS

  basicAccessControl, id-at, id-ce, id-mr, informationFramework,
  authenticationFramework, selectedAttributeTypes, id-oc, certificateExtensions
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 7}

  ATTRIBUTE, Attribute{}, AttributeType, MATCHING-RULE, Name, OBJECT-CLASS,
  RelativeDistinguishedName, SupportedAttributes, top
    FROM InformationFramework informationFramework

  AttributeTypeAndValue
    FROM BasicAccessControl basicAccessControl

  AlgorithmIdentifier, Certificate, CertificateList, CertificateSerialNumber,
    EXTENSION, Extensions, InfoSyntax, PolicySyntax, SIGNED{}, SupportedAlgorithms
    FROM AuthenticationFramework authenticationFramework

  TimeSpecification, UnboundedDirectoryString, UniqueIdentifier
    FROM SelectedAttributeTypes selectedAttributeTypes

  certificateListExactMatch, GeneralName, GeneralNames, NameConstraintsSyntax
    FROM CertificateExtensions certificateExtensions

  UserNotice
    FROM PKIX1Implicit93 {iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-implicit-93(4)};

-- 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{AttributeCertificateInfo}

AttributeCertificateInfo ::= 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 }

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-targetInformation }

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-allowedAttAss }

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
  ID                       id-at-attributeCertificateRevocationList }

attributeAuthorityRevocationList ATTRIBUTE ::= {
  WITH SYNTAX              CertificateList
  EQUALITY MATCHING RULE   certificateListExactMatch
  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}

-- 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-targetInformation                 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-allowedAttAss                     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 - Password policy module

PasswordPolicy {joint-iso-itu-t ds(5) module(1) passwordPolicy(39) 7}
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
  authenticationFramework, id-asx, id-at, id-mr, id-oa, informationFramework,
  selectedAttributeTypes
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1)
     usefulDefinitions(0) 7}

  AlgorithmIdentifier{}, ALGORITHM, EXTENSION, SupportedAlgorithms
    FROM AuthenticationFramework authenticationFramework

  ATTRIBUTE, MATCHING-RULE, pwdHistory{}, pwdRecentlyExpired{},
  pwdHistoryMatch{}, SYNTAX-NAME
    FROM InformationFramework informationFramework

  bitStringMatch, boolean, booleanMatch, directoryString, generalizedTime,
  generalizedTimeMatch,
  generalizedTimeOrderingMatch, integer, integerMatch, integerOrderingMatch, uri
    FROM SelectedAttributeTypes selectedAttributeTypes;

userPwd    ATTRIBUTE ::= {
  WITH SYNTAX               UserPwd
  EQUALITY MATCHING RULE    userPwdMatch
  SINGLE VALUE              TRUE
  LDAP-SYNTAX               userPwdDescription.&id
  LDAP-NAME                 {"userPwd"}
  ID                        id-at-userPwd }

UserPwd ::= CHOICE {
  clear                  UTF8String,
  encrypted              SEQUENCE {
    algorithmIdentifier    AlgorithmIdentifier{{SupportedAlgorithms}},
    encryptedString        OCTET STRING,
    ...},
  ...}

-- Operational attributes

pwdStartTime ATTRIBUTE ::= {
  WITH SYNTAX               GeneralizedTime
  EQUALITY MATCHING RULE    generalizedTimeMatch
  ORDERING MATCHING RULE    generalizedTimeOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               generalizedTime.&id
  LDAP-NAME                 {"pwdStartTime"}
  ID                        id-oa-pwdStartTime }

pwdExpiryTime ATTRIBUTE ::= {
  WITH SYNTAX               GeneralizedTime
  EQUALITY MATCHING RULE    generalizedTimeMatch
  ORDERING MATCHING RULE    generalizedTimeOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               generalizedTime.&id
  LDAP-NAME                 {"pwdExpiryTime"}
  ID                        id-oa-pwdExpiryTime }

pwdEndTime ATTRIBUTE ::= {
  WITH SYNTAX               GeneralizedTime
  EQUALITY MATCHING RULE    generalizedTimeMatch
  ORDERING MATCHING RULE    generalizedTimeOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               generalizedTime.&id
```

```
  LDAP-NAME                 {"pwdEndTime"}
  ID                        id-oa-pwdEndTime }

pwdFails ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (0..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     dSAOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdFails"}
  ID                        id-oa-pwdFails }

pwdFailureTime ATTRIBUTE ::= {
  WITH SYNTAX               GeneralizedTime
  EQUALITY MATCHING RULE    generalizedTimeMatch
  ORDERING MATCHING RULE    generalizedTimeOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     dSAOperation
  LDAP-SYNTAX               generalizedTime.&id
  LDAP-NAME                 {"pwdFailureTime"}
  ID                        id-oa-pwdFailureTime }

pwdGracesUsed ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (0..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     dSAOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdGracesUsed"}
  ID                        id-oa-pwdGracesUsed }

userPwdHistory ATTRIBUTE ::=
                    pwdHistory{userPwd,userPwdHistoryMatch,id-oa-userPwdHistory}

userPwdRecentlyExpired ATTRIBUTE ::=
                    pwdRecentlyExpired{userPwd,id-oa-userPwdRecentlyExpired}

pwdModifyEntryAllowed ATTRIBUTE ::= {
  WITH SYNTAX               BOOLEAN
  EQUALITY MATCHING RULE    booleanMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               boolean.&id
  LDAP-NAME                 {"pwdModifyEntryAllowed"}
  ID                        id-oa-pwdModifyEntryAllowed }

pwdChangeAllowed ATTRIBUTE ::= {
  WITH SYNTAX               BOOLEAN
  EQUALITY MATCHING RULE    booleanMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               boolean.&id
  LDAP-NAME                 {"pwdChangeAllowed"}
  ID                        id-oa-pwdChangeAllowed }

pwdMaxAge ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (1 .. MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdMaxAge"}
  ID                        id-oa-pwdMaxAge }

pwdExpiryAge ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (1 .. MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
```

```
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdExpiryAge"}
  ID                        id-oa-pwdExpiryAge }

pwdMinLength ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (0..MAX)
  EQUALITY MATCHING RULE    integerMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdMinLength"}
  ID                        id-oa-pwdMinLength }

pwdVocabulary ATTRIBUTE ::= {
  WITH SYNTAX               PwdVocabulary
  EQUALITY MATCHING RULE    bitStringMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               pwdVocabularyDescription.&id
  LDAP-NAME                 {"pwdVocabulary"}
  ID                        id-oa-pwdVocabulary }

PwdVocabulary ::=  BIT STRING {
    noDictionaryWords   (0),
    noPersonNames       (1),
    noGeographicalNames (2) }

pwdAlphabet ATTRIBUTE ::= {
  WITH SYNTAX               PwdAlphabet
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               pwdAlphabetDescription.&id
  LDAP-NAME                 {"pwdAlphabet"}
  ID                        id-oa-pwdAlphabet }

PwdAlphabet ::= SEQUENCE OF UTF8String

pwdDictionaries ATTRIBUTE ::= {
  SUBTYPE OF                uri
  USAGE                     directoryOperation
  LDAP-SYNTAX               directoryString.&id
  LDAP-NAME                 {"pwdDictionaries"}
  ID                        id-oa-pwdDictionaries }

pwdExpiryWarning ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (1..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdExpiryWarning"}
  ID                        id-oa-pwdExpiryWarning }

pwdGraces ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (0..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdGraces"}
  ID                        id-oa-pwdGraces }

pwdFailureDuration ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (0..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
```

```
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdFailureDuration"}
  ID                        id-oa-pwdFailureDuration }

pwdLockoutDuration ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (0..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdLockoutDuration"}
  ID                        id-oa-pwdLockoutDuration }

pwdMaxFailures ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (1..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdMaxFailures"}
  ID                        id-oa-pwdMaxFailures }

pwdMaxTimeInHistory ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (1..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdMaxTimeInHistory"}
  ID                        id-oa-pwdMaxTimeInHistory }

pwdMinTimeInHistory ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (0..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdMinTimeInHistory"}
  ID                        id-oa-pwdMinTimeInHistory }

pwdHistorySlots ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (2..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdHistorySlots"}
  ID                        id-oa-pwdHistorySlots }

pwdRecentlyExpiredDuration ATTRIBUTE ::= {
  WITH SYNTAX               INTEGER (0..MAX)
  EQUALITY MATCHING RULE    integerMatch
  ORDERING MATCHING RULE    integerOrderingMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               integer.&id
  LDAP-NAME                 {"pwdRecentlyExpiredDuration"}
  ID                        id-oa-pwdRecentlyExpiredDuration }

pwdEncAlg ATTRIBUTE ::= {
  WITH SYNTAX               PwdEncAlg
  EQUALITY MATCHING RULE    pwdEncAlgMatch
  SINGLE VALUE              TRUE
  USAGE                     directoryOperation
  LDAP-SYNTAX               pwdEncAlgDescription.&id
```

```
  LDAP-NAME                {"pwdEncAlg"}
  ID                       id-oa-pwdEncAlg }

PwdEncAlg ::= AlgorithmIdentifier{{SupportedAlgorithms}}

userPwdMatch MATCHING-RULE ::= {
  SYNTAX       UserPwd
  LDAP-SYNTAX  userPwdDescription.&id
  LDAP-NAME    {"userPwdMatch"}
  ID           id-mr-userPwdMatch }

pwdEncAlgMatch MATCHING-RULE ::= {
  SYNTAX       PwdEncAlg
  LDAP-SYNTAX  pwdEncAlgDescription.&id
  LDAP-NAME    {"pwdEncAlgMatch"}
  ID           id-mr-pwdEncAlgMatch }

userPwdHistoryMatch MATCHING-RULE ::= pwdHistoryMatch{userPwd,id-mr-userPwdHistoryMatch}

-- LDAP syntaxes defined by this Directory Specification

userPwdDescription SYNTAX-NAME ::= {
  LDAP-DESC          "User Password Description"
  DIRECTORY SYNTAX   UserPwd
  ID                 id-asx-userPwdDescription }

pwdVocabularyDescription SYNTAX-NAME ::= {
  LDAP-DESC          "Password Vocabulary Description"
  DIRECTORY SYNTAX   PwdVocabulary
  ID                 id-asx-pwdVocabularyDescription }

pwdAlphabetDescription SYNTAX-NAME ::= {
  LDAP-DESC          "Password Alphabet Description"
  DIRECTORY SYNTAX   PwdAlphabet
  ID                 id-asx-pwdAlphabetDescription }

pwdEncAlgDescription SYNTAX-NAME ::= {
  LDAP-DESC          "Password Alphabet Description"
  DIRECTORY SYNTAX   PwdEncAlg
  ID                 id-asx-pwdEncAlgDescription }

-- object identifier assignments

-- directory attributes

id-at-userPwd                    OBJECT IDENTIFIER ::= {id-at 85}

-- operational attributes --

id-oa-pwdStartTime               OBJECT IDENTIFIER ::= {id-oa 22}
id-oa-pwdExpiryTime              OBJECT IDENTIFIER ::= {id-oa 23}
id-oa-pwdEndTime                 OBJECT IDENTIFIER ::= {id-oa 24}
id-oa-pwdFails                   OBJECT IDENTIFIER ::= {id-oa 25}
id-oa-pwdFailureTime             OBJECT IDENTIFIER ::= {id-oa 26}
id-oa-pwdGracesUsed              OBJECT IDENTIFIER ::= {id-oa 27}
id-oa-userPwdHistory             OBJECT IDENTIFIER ::= {id-oa 28}
id-oa-userPwdRecentlyExpired     OBJECT IDENTIFIER ::= {id-oa 29}
id-oa-pwdModifyEntryAllowed      OBJECT IDENTIFIER ::= {id-oa 30}
id-oa-pwdChangeAllowed           OBJECT IDENTIFIER ::= {id-oa 31}
id-oa-pwdMaxAge                  OBJECT IDENTIFIER ::= {id-oa 32}
id-oa-pwdExpiryAge               OBJECT IDENTIFIER ::= {id-oa 33}
id-oa-pwdMinLength               OBJECT IDENTIFIER ::= {id-oa 34}
id-oa-pwdVocabulary              OBJECT IDENTIFIER ::= {id-oa 35}
id-oa-pwdAlphabet                OBJECT IDENTIFIER ::= {id-oa 36}
id-oa-pwdDictionaries            OBJECT IDENTIFIER ::= {id-oa 37}
id-oa-pwdExpiryWarning           OBJECT IDENTIFIER ::= {id-oa 38}
id-oa-pwdGraces                  OBJECT IDENTIFIER ::= {id-oa 39}
id-oa-pwdFailureDuration         OBJECT IDENTIFIER ::= {id-oa 40}
id-oa-pwdLockoutDuration         OBJECT IDENTIFIER ::= {id-oa 41}
id-oa-pwdMaxFailures             OBJECT IDENTIFIER ::= {id-oa 42}
id-oa-pwdMaxTimeInHistory        OBJECT IDENTIFIER ::= {id-oa 43}
```

```
id-oa-pwdMinTimeInHistory         OBJECT IDENTIFIER ::= {id-oa 44}
id-oa-pwdHistorySlots             OBJECT IDENTIFIER ::= {id-oa 45}
id-oa-pwdRecentlyExpiredDuration  OBJECT IDENTIFIER ::= {id-oa 46}
id-oa-pwdEncAlg                   OBJECT IDENTIFIER ::= {id-oa 47}

-- matching rules

id-mr-userPwdMatch                OBJECT IDENTIFIER ::= {id-mr 71}
id-mr-userPwdHistoryMatch         OBJECT IDENTIFIER ::= {id-mr 72}
id-mr-pwdEncAlgMatch              OBJECT IDENTIFIER ::= {id-mr 73}

-- syntaxes

id-asx-userPwdDescription         OBJECT IDENTIFIER ::= {id-asx 0}
id-asx-pwdVocabularyDescription   OBJECT IDENTIFIER ::= {id-asx 1}
id-asx-pwdAlphabetDescription     OBJECT IDENTIFIER ::= {id-asx 2}
id-asx-pwdEncAlgDescription       OBJECT IDENTIFIER ::= {id-asx 3}

END -- Password policy
```

Annex B

Reference definition of algorithm object identifiers

(This annex forms an integral part of this Recommendation | International Standard.)

This annex defines object identifiers assigned to authentication and encryption algorithms, in the absence of a formal register. It is intended to make use of such a register as it becomes available. The definitions take the form of the ASN.1 module, **AlgorithmObjectIdentifiers**.

```
AlgorithmObjectIdentifiers {joint-iso-itu-t ds(5) module(1)
  algorithmObjectIdentifiers(8) 7}
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
  algorithm, authenticationFramework
    FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 7}

  ALGORITHM
    FROM AuthenticationFramework authenticationFramework;

-- categories of object identifier

nullAlgorithm        OBJECT IDENTIFIER ::= {algorithm 0}
encryptionAlgorithm  OBJECT IDENTIFIER ::= {algorithm 1}
hashAlgorithm        OBJECT IDENTIFIER ::= {algorithm 2}
signatureAlgorithm   OBJECT IDENTIFIER ::= {algorithm 3}

-- synonyms

id-ea                OBJECT IDENTIFIER ::= encryptionAlgorithm
id-ha                OBJECT IDENTIFIER ::= hashAlgorithm
id-sa                OBJECT IDENTIFIER ::= signatureAlgorithm

-- algorithms

rsa ALGORITHM ::= {
                  KeySize
  IDENTIFIED BY  id-ea-rsa
}

KeySize ::= INTEGER

-- the following object identifier assignments reserve values assigned to deprecated
functions

id-ea-rsa            OBJECT IDENTIFIER ::= {id-ea 1}
id-ha-sqMod-n        OBJECT IDENTIFIER ::= {id-ha 1}
id-sa-sqMod-nWithRSA OBJECT IDENTIFIER ::= {id-sa 1}

-- the following object identifier are related to password hashing methods

mD5Algorithm ALGORITHM ::= {
                  NULL
  IDENTIFIED BY {iso(1) member-body(2) us(840) rsadsi(113549) digestAlgorithm(2) md5(5)}}

sha1Algorithm ALGORITHM ::= {
                  NULL
  IDENTIFIED BY {iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 26}}

END -- AlgorithmObjectIdentifiers
```

Annex C

CRL generation and processing rules

(This annex forms an integral part of this Recommendation | International Standard.)

C.1 Introduction

A relying party needs the ability to check the revocation status of a certificate in order to determine whether or not to trust that certificate. Certificate Revocation Lists (CRL) are one mechanism for relying parties to obtain the revocation information. Other mechanisms may also be used, but are outside the scope of this Directory Specification.

This annex addresses the use of CRLs for certificate revocation status checking by relying parties. Various authorities may have different policies regarding their issuance of revocation lists. For instance, in some cases the certificate issuing authority may authorize a different authority to issue a certificate revocation list for the certificates it issues. Some authorities may combine the revocation of end-entity and CA-certificates into a single list while other authorities may split these into separate lists. Some authorities may partition their certificate population onto CRL fragments and some authorities may issue delta updates to a revocation list between regular CRL intervals. As a result, relying parties need to be able to determine the scope of the CRLs they retrieve to enable them to ensure they have the complete set of revocation information covering the scope of the certificate in question for the revocation reasons of interest, given the policy under which they are working. This annex provides a mechanism for the relying parties to determine the scope of retrieved CRLs.

This annex is written for revocation status checking of public-key certificates using CRLs, Full and Complete End-Entity CRLs (EPRLs) and CA Revocation Lists (CARLs). However, this description can also be applied to revocation status checking of attribute certificates using Attribute Certificate Revocation Lists (ACRL) and Attribute Authority Revocation Lists (AARL). For the purposes of this annex, ACRL can be considered in place of CRL, EPRL can be full and complete end-entity ACRL, and AARL in place of CARL. Similarly, the directory attributes identified in clause C.4 shall be mapped to those for the AARL and ACRL and the fields identifying certificate types in the Issuing Distribution Point extension can be mapped to those applicable to PMI.

C.1.1 CRL types

CRLs of one or more of the following types may be available to a relying party, based on the revocation aspects of the policy of the certificate issuing authority:

- Full and complete CRL;
- Full and complete end-entity CRL (EPRL);
- Full and complete CA Revocation List (CARL);
- Distribution Point CRL, EPRL or CARL;
- Indirect CRL, EPRL or CARL (ICRL);
- Delta CRL, EPRL or CARL;
- Indirect dCRL, EPRL or CARL.

A full and complete CRL is a list of all revoked end-entity and CA-certificates issued by an authority for any and all reasons.

A full and complete EPRL is a list of all revoked end-entity certificates issued by an authority for any and all reasons.

A full and complete CARL is a list of revoked CA-certificates issued by an authority for any and all reasons.

A distribution point CRL, EPRL or CARL is one that covers all or a subset of certificates issued by an authority. The subset could be based on a variety of criteria.

An indirect CRL, EPRL or CARL (ICRL) is a CRL that contains a list of revoked certificates, in which some or all of those certificates were not issued by the authority signing and issuing the CRL.

A delta CRL, EPRL or CARL is a CRL that only contains changes to a CRL that is complete for the given scope at the time of the CRL referenced in the dCRL. Note that the referenced CRL might be one that is complete for the given scope or it might be a dCRL that is used to locally construct a CRL that is complete for the given scope.

All of the above CRL types (except for the dCRL) are CRL types that are complete for their given scope. A dCRL shall be used in conjunction with an associated CRL that is complete for the same scope in order to form a complete picture of the revocation status of certificates.