TELECOMMUNICATION STANDARDIZATION SECTOR

STUDY PERIOD 1997 - 2000

INTERNATIONAL TELECOMMUNICATION UNION

COM 7-250-E Revision 1
Feb 23, 2001
Original: English

TITLE: ITU-T RECOMMENDATION X.509 | ISO/IEC 9594-8: "INFORMATION TECHNOLOGY – OPEN SYSTEMS INTERCONNECTION – THE DIRECTORY: PUBLIC-KEY AND ATTRIBUTE CERTIFICATE FRAMEWORKS"

Summary

This Recommendation | International Standard also defines a framework for public-key certificates and attribute certificates. These frameworks may be used by other standards bodies to profile their application to Public Key Infrastructures (PKI) and Privilege Management Infrastructures (PMI). Also, this Recommendation | International Standard defines a framework for the provision of authentication services by Directory to its users. It 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.

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. That framework includes 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 Specification, while it defines some critical components of a Public-key Infrastructure (PKI), it does not define a PKI in its entirety. However, this Specification 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 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 Specification, while it defines some critical components of a Privilege Management Infrastructure (PMI), it does not define a PMI it its entirety. However, this Specification 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 fourth edition technically revises and enhances, but does not replace, the third edition of this Recommendation | International Standard. Implementations may still claim conformance to the third edition. However, at some point, the third edition will not be supported (i.e. reported defects will no longer be resolved). It is recommended that implementations conform to this fourth edition as soon as possible.

This fourth edition specifies version 1 and version 2 of the Directory protocols.

The first and second editions specified only version 1. Most of the services and protocols specified in this edition are designed to function under version 1. However some enhanced services and protocols, e.g. signed errors, will not function unless all Directory entities involved in the operation have negotiated version 2. Whichever version has been negotiated, differences between the services and between the protocols defined in the four editions, except for those specifically assigned to version 2, are accommodated using the rules of extensibility defined in this edition of ITU-T Rec. X.519 | ISO/IEC 9594-5.

Attention is drawn to the existence of a "defect resolution" procedure in ISO/IEC JTC 1 to identify and correct errors in International Standards through the publication of Technical Corrigenda. Identical corrections are made to the corresponding ITU-T Recommendations through Corrigenda and may also be made in the form of Implementors' Guides. Details of

Technical Corrigenda to International Standards are available on the ISO website; published Technical Corrigenda can be obtained via the ISO webstore or from the ISO and IEC national bodies. Corrigenda and Implementors' Guides to ITU-T Recommendations can be obtained from the ITU-T website.

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

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

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

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

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

Annex F, 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 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, contains an alphabetical list of information item definitions in this Specification.

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

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – OPEN SYSTEMS INTERCONNECTION – THE DIRECTORY: PUBLIC-KEY AND ATTRIBUTE CERTIFICATE FRAMEWORKS

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;
- Authentication services.

The public-key certificate framework defined in this Recommendation | International Standard includes definition of the information objects for Public Key Infrastructure (PKI), including public-key certificates and Certificate Revocation List (CRLs). The attribute certificate framework includes definition of the information objects for Privilege Management Infrastructure (PMI), including attribute certificates, and Attribute Certificate Revocation List (ACRL). This Specification 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 uses of public-key certificates and attribute certificates and the framework for 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, signed and/or encrypted operations, and for storage of signed and/or encrypted 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 Specification, 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 Directory Specifications.

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 ITU-T Rec. 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

– ITU-T Recommendation X.411 | ISO/IEC 10021-4, Information technology — Message Handling Systems — Message Transfer System: Abstract service definition and procedures.

– ITU-T Recommendation X.500 (2001) | ISO/IEC 9594-1:2001, Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services.[1]

– ITU-T Recommendation X.501 (2001) | ISO/IEC 9594-2:2001, Information technology – Open Systems Interconnection – The Directory: Models.

– ITU-T Recommendation X.511 (2001) | ISO/IEC 9594-3:2001, Information technology – Open Systems Interconnection – The Directory: Abstract service definition.

– ITU-T Recommendation X.518 (2001) | ISO/IEC 9594-4:2001, Information technology – Open Systems Interconnection – The Directory: Procedures for distributed operation.

– ITU-T Recommendation X.519 (2001) | ISO/IEC 9594-5:2001, Information technology – Open Systems Interconnection – The Directory: Protocol specifications.

– ITU-T Recommendation X.520 (2001) | ISO/IEC 9594-6:2001, Information technology – Open Systems Interconnection – The Directory: Selected attribute types.

– ITU-T Recommendation X.521 (2001) | ISO/IEC 9594-7:2001, Information technology – Open Systems Interconnection – The Directory: Selected object classes.

– ITU-T Recommendation X.525 (2001) | ISO/IEC 9594-9:2001, Information technology – Open Systems Interconnection – The Directory: Replication.

---

[1] For each of the X.500 series of Recommendations | 9594 parts referenced in this clause, the 4th edition of those Specifications should be used when they become available.

– ITU-T Recommendation X.530 (2001) | ISO/IEC 9594-10:2001, Information technology – Open Systems Interconnection – The Directory: Use of Systems Managemnt for administration of the Directory

– ITU-T Rec. X.660 (1992) | ISO/IEC 9834-1:1993, Information technology – Open Systems Interconnection Procedures for the operation of OSI Registration Authorities: General procedures.

– X.680 : ITU-T Recommendation X.680 (1997) | ISO/IEC 8824-1:1998, Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation

– X.681 : ITU-T Recommendation X.681 (1997) | ISO/IEC 8824-2:1998, Information Technology - Abstract Syntax Notation One (ASN.1): Information Object Specification

– X.682 : ITU-T Recommendation X.682 (1997) | ISO/IEC 8824-3:1998, Information Technology - Abstract Syntax Notation One (ASN.1): Constraint Specification

– X.683 : ITU-T Recommendation X.683 (1997) | ISO/IEC 8824-4:1998, Information Technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 Specifications

– X.690 : ITU-T Recommendation X.690 (1997) | ISO/IEC 8825-1:1998, Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

– X.691 : ITU-T Recommendation X.691 (1997) | ISO/IEC 8825-2:1998, Information Technology - ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER)

– ITU-T Recommendation X.812 (1995) | ISO/IEC 10181-3:1996 Information technology - Open Systems Interconnection – Security frameworks for open systems: Access control framework

– ITU-T Recommendation X.813 (1995) | ISO/IEC 10181-4:1996, Information Technology – Open Systems Interconnection – Security frameworks for open systems: Non-repudiation framework

– ITU-T Recommendation X.880 (1994) | ISO/IEC 13712-1:1994, Information technology – Remote Operations: Concepts, model and notation.

– ITU-T Recommendation X.881 (1994) | ISO/IEC 13712-2:1994, Information technology – Remote Operations: OSI realizations – Remote Operations Service Element (ROSE) service definition.

2.2 Paired Recommendations | International Standards equivalent in technical content

– CCITT Recommendation 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.

3 Definitions

For the purposes of this ITU-T 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) encipherment;
j) key;
k) password;
l) peer-entity authentication;
m) symmetric (encipherment).

3.2 Directory model definitions

The following terms are defined in ITU-T Rec. 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.

3.3 Definitions

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

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

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

3.3.3 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.

3.3.4 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.

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

3.3.6 authority: An entity, responsible for the issuance of certificates. Two types are defined in this Specification; certification authority which issues public-key certificates and attribute authority which issues attribute certificates.

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

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

3.3.9 CA-certificate: A certificate for one CA issued by another CA.

3.3.10 **certificate policy***:* A named set of rules that indicates 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 applicability of a type of certificate to the authentication of electronic data interchange transactions for the trading of goods within a given price range.

3.3.11 **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.

3.3.12 **certificate user**: An entity that needs to know, with certainty, the public key of another entity.

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

3.3.14 **certificate-using system**: An implementation of those functions defined in this Directory Specification that are used by a certificate-user.

3.3.15 **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.

3.3.16 **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 users' keys.

3.3.17 **Certification Authority Revocation List (CARL):** A revocation list containing a list of public-key certificates issued to certification authorities, that are no longer considered valid by the certificate issuer.

3.3.18 **certification path**: An ordered sequence of certificates of objects in the DIT which, together with the public key of the initial object in the path, can be processed to obtain that of the final object in the path.

3.3.19 **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.

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

3.3.21 **data confidentiality**: This service can be used to provide for 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.

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

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

3.3.24 **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.

3.3.25 **end entity***:* A certificate subject that uses its private key for purposes other than signing certificates or an entity that is a relying party.

3.3.26 **End-entity Attribute Certificate Revocation List (EARL):** A revocation list containing a list of attribute certificates issued to holders, that are not also AAs, that are no longer considered valid by the certificate issuer.

3.3.27 **End-entity Public-key Certificate Revocation List (EPRL):** A revocation list containing a list of public-key certificates issued to subjects, that are not also CAs, that are no longer considered valid by the certificate issuer.

3.3.28 **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).

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

3.3.30 **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.

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

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

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

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

3.3.35 **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 y for which finding x is not computationally difficult.

3.3.36 **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.

3.3.37 **private key; secret key** (deprecated): (In a public key cryptosystem) that key of a user's key pair which is known only by that user.

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

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

3.3.40 **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.

3.3.41 **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.

3.3.42 **privilege verifier:** An entity verifying certificates against a privilege policy.

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

3.3.44 **public-key certificate**: The public key of a user, together with some other information, rendered unforgeable by encipherment with the private key of the certification authority which issued it.

3.3.45 **Public Key Infrastructure (PKI):** The infrastructure able to support the management of public keys able to support authentication, encryption, integrity or non-repudiation services.

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

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

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

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

3.3.50 **simple authentication**: Authentication by means of simple password arrangements.

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

3.3.52 **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.

3.3.53 **strong authentication**: Authentication by means of cryptographically derived credentials.

3.3.54 **trust**: Generally, an entity can be said to "trust" a second entity when it (the first entity) makes the assumption that the second entity will behave exactly as the first entity expects. This trust may apply only for some specific function. The key role of trust in this framework is to describe the relationship between an authenticating entity and a authority; an entity shall be certain that it can trust the authority to create only valid and reliable certificates.

4 Abbreviations

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

AA Attribute Authority
AARL Attribute Authority Revocation List
ACRL Attribute Certificate Revocation List
CA Certification Authority
CARL Certification Authority Revocation List
CRL Certificate Revocation List
dCRL Delta Certificate Revocation List
DIB Directory Information Base
DIT Directory Information Tree
DSA Directory System Agent
DUA Directory User Agent
EARL End-entity Attribute certificate Revocation List
EPRL End-entity Public-key certificate Revocation List
iCRL Indirect Certificate Revocation List
PKCS Public Key Cryptosystem
PKI Public-Key Infrastructure
PMI Privilege Management Infrastructure
SOA Source of Authority

5 Conventions

With minor exceptions this Directory Specification has been prepared according to the "Presentation of ITU-TS/ISO/IEC common text" guidelines in the Guide for ITU-TS and ISO/IEC JTC 1 Cooperation, March 1993.

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

This Directory Specification uses the term "1988 edition systems" to refer to systems conforming to the first edition of the 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 "1993 edition systems" to refer to systems conforming to the second (1993) edition of the 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 "1997 edition systems" to refer to systems conforming to the third edition of the 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 "4th edition systems" to refer to systems conforming to this fourth edition of the Directory Specifications ", i.e. the 2001 editions of 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 ITU-T X.509, parts 1–7 and parts 9–10 of the ISO/IEC 9594:2001 edition, and ISO/IEC 9594-8:2000.

This Directory Specification presents ASN.1 notation in the bold Helvetica typeface. When ASN.1 types and values are referenced in normal text, they are differentiated from normal text by presenting them in the bold Helvetica typeface. The names of procedures, typically referenced when specifying the semantics of processing, are differentiated from normal text by displaying them in bold Times. Access control permissions are presented in italized Times.

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

The notation used in this Directory Specification is defined in Table 1 below.

Table 1 – Notation

Notation	Meaning
Xp	Public key of a user X.
Xs	Private key of X.
Xp[I]	Encipherment of some information, I, using the public key of X.
Xs[I]	Encipherment of I using the private key of X.
X{I}	The signing of I by user X. It consists of I with an enciphered summary appended.
CA(X)	A certification authority of user X.
$CA^n(X)$	(Where n>1): CA(CA(...n times...(X)))
$X_1«X_2»$	The certificate of user X_2 issued by certification authority X_1.
$X_1«X_2»\ X_2«X_3»$	A chain of certificates (can be of arbitrary length), where each item is the certificate for the certification authority which produced the next. It is functionally equivalent to the following certificate $X_1«X_{n+1}»$. For example, possession of A«B»B«C» provides the same capability as A«C», namely the ability to find out Cp given Ap.
$X_1p \bullet X_1«X_2»$	The operation of unwrapping a certificate (or certificate chain) to extract a public key. It is an infix operator, whose left operand is the public key of a certification authority, and whose right operand is a certificate issued by that certification authority. The outcome is the public key of the user whose certificate is the right operand. For example: Ap • A«B» B«C» denotes the operation of using the public key of A to obtain B's public key, Bp, from its certificate, followed by using Bp to unwrap C's certificate. The outcome of the operation is the public key of C, Cp.
A→B	A certification path from A to B, formed of a chain of certificates, starting with $CA(A)«CA^2(A)»$ and ending with CA(B)«B».
NOTE — In the table, the symbols X, X_1, X_2, etc., occur in place of the names of users, while the symbol I occurs in place of arbitrary information.	

6 Frameworks overview

This 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 certificate by the certificate user. The validation includes:

- establishing a trusted path of certificates between the certificate user and the certificate subject;
- verifying the digital signatures on each certificate in the path; and
- validating all the certificates along that path (i.e. that they were not expired or not revoked at a given time).

This Specification defines a framework for obtaining and trusting privilege attributes of an entity in order to determine whether or not they are authorized to access a particular resource. The framework includes the issuance of a certificate by an Attribute Authority (AA) and the validation of that 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 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.

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 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 Specification discuss the use of digital signatures within PKI and PMI specifically.

This clause is not intended to specify a standard for digital signatures in general, but to specify the means by which the tokens are signed in PKI, PMI and in the Directory.

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:

$$X\{Info\} = Info, Xs[h\,(Info)]$$

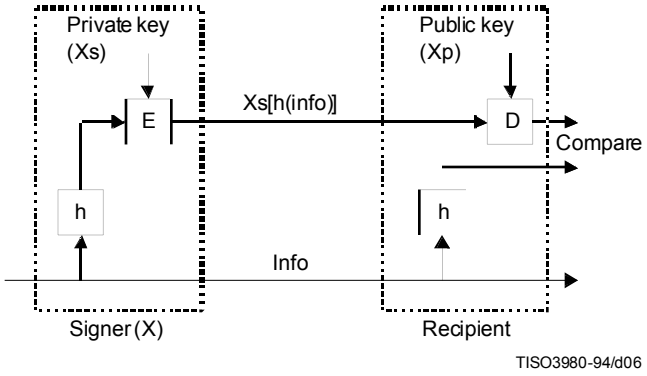


Figure 1 – Digital signatures

NOTE 1 – 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 so as 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 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 as a result 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.

The signed information includes indicators that identify the hashing algorithm and the encryption algorithm used to compute the digital signature.

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

```
ENCRYPTED { ToBeEnciphered }    ::=        BIT STRING ( CONSTRAINED BY {
  -- must 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 – ITU-T Rec. X.690 (1997) | SO/IEC 8825-1:1998) of the value of the **ToBeEnciphered** type and applying an encipherment procedure to those octets.

NOTE 2 — 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 3 — 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 4 — 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 some 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,
  hashValue                        BIT STRING ( CONSTRAINED BY {
        -- must 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 {
  -- must be the result of applying a hashing procedure to the DER-encoded octets --
  -- of a value of -- ToBeSigned -- and then applying an encipherment procedure to those octets -- })

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

NOTE 5 — The encryption procedure requires the agreements listed in Note 2, and also 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 must be 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 }}
```

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 ITU-T Rec. X.690 (1997) | ISO/IEC 8825 :1998, 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"16;

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 ITU-T Rec. X.690 (1997) | ISO/IEC 8825-1:1998;

j) the encoding of a Generalized time shall be as specified in ITU-T Rec. X.690 (1997) | ISO/IEC 8825-1:1998.

Generating a distinguished encoding requires the abstract syntax of the data to be encoded to be fully understood. The Directory may be required to sign data or check the signature of data that contains unknown protocol extensions or unknown attribute syntaxes. The Directory 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 public key 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.

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

– strong authentication between and among directory components;

– authentication, integrity and confidentiality of directory operations; as well as

– integrity and authentication of stored data.

7 Public-keys and public-key certificates

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 must 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 the possession of the private key and other procedures

related to the issuance of public-key certificates are outside the scope of this 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 certification authority can recover the public key which was certified;
- no party other than the certification authority 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 1 — 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.

A certification authority produces the certificate of a user by signing (see subclause 6.1) a collection of information, including the user's distinguished name and public key, as well as an optional *unique identifier* containing additional information about the user. The exact form of the unique identifier contents is unspecified here and left to the certification authority and might be, for example, an object identifier, a certificate, a date, or some other form of certification on the validity of the distinguished name. Specifically, the certificate of a user with distinguished name A and unique identifier UA, produced by the certification authority with name CA and unique identifier UCA, has the following form:

$$CA\langle\langle A\rangle\rangle = CA\{V,SN,AI,CA,UCA,A,UA,Ap,T^A\}$$

where V is the version of the certificate, SN is the serial number of the certificate, AI is the identifier of the algorithm used to sign the certificate, UCA is the optional unique identifier of the CA, UA is the optional unique identifier of the user A, T^A indicates the period of validity of the certificate, and consists of two dates, the first and last on which the certificate is valid. The certificate validity period is the time interval during which the CA warrants that it will maintain information about the status of the certificate, i.e. publish revocation data. Since T^A is assumed to be changed in periods not less than 24 hours, it is expected that systems would use Coordinated Universal Time as a reference time base. The signature in the certificate can be checked for validity by any user with knowledge of CAp. The following ASN.1 data type can be used to represent certificates:

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


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

CertificateSerialNumber     ::=     INTEGER

AlgorithmIdentifier         ::=     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       ::=     { ... }
```

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

SubjectPublicKeyInfo              ::=       SEQUENCE {
  algorithm                   AlgorithmIdentifier,
  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
               -- contains a DER encoding of a value of type &ExtnType
               -- for the extension object identified by extnId -- }

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 49 inclusive, the value shall have 2000 added to it.

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

NOTE 2 — 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.

version is the version of the encoded certificate. If the **extensions** component is present in the certificate, version shall be v3. If the **issuerUniqueIdentifier** or **subjectUniqueIdentifier** component is present version must be v2 or v3.

If unkown 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 7.5.2.2 in ITU-T Rec. X.519 | ISO/IEC 9594-5.

serialNumber is an integer assigned by the CA to each certificate. The value of **serialNumber** must be unique for each certificate issued by a given CA (i.e., the issuer name and serial number identify a unique certificate).

signature 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.)

issuer identifies the entity that has signed and issued the certificate.

validity is the time interval during which the CA warrants that it will maintain information about the status of the certificate.

subject identifies the entity associated with the public-key found in the subject public key field.

subjectPublicKeyInfo is used to carry the public key being certified and to identify the algorithm which this public key is an instance of (e.g. rsaEncryption, dhpublicnumber, id-dsa, etc.)

issuerUniqueIdentifier is used to uniquely identify an issuer in case of name re-use.

subjectUniqueIdentifier is used to uniquely identify a subject in case of name re-use.

NOTE 3 — In situations where a distinguished name might be reassigned to a different user by the Naming Authority, CAs can use the unique identifier to distinguish between reused instances. However, if the same user is provided certificates by multiple

CAs, it is recommended that the CAs coordinate on the assignment of unique identifiers as part of their user registration procedures.

The **extensions** field allows 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 an implementation processing a certificate does not recognize an extension, if 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 certificate to fail. When a certificate-using implementation recognizes and is able to process an extension, then the certificate-using implementation shall process the extension regardless of the value of the criticality flag. Note that any extension that is flagged non-critical will cause inconsistent behaviour between certificate-using systems that will process the extension and certificate-using systems that do not recognize the extension and will ignore it.

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 extensiblity documented in 7.5.2.2 in ITU-T Rec. X.519 | ISO/IEC 9594-5.

A CA 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 validation engine has two possible actions to take with respect to an extension:

i) it can ignore the extension and accept the certificate (all other things being equal);

ii) it can process the extension and accept or reject the certificate depending on the content of the extension and the conditions under which processing is occuring (e.g. the current values of the path processing variables).

Some extensions can only be marked critical. In these cases a validation engine that understands the extension, processes it and acceptance/rejection of the certificate is dependent (at least in part) on the content of the extension. A validation engine that does not understand the extension rejects the certificate.

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

Some extensions can be marked critical or non-critical. In these cases a validation engine that understands the extension processes it and acceptance/rejection of the certificate is dependent (at least in part) on the content of the extension, regardless of the criticality flag. A validation engine 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 certificate, a CA would flag the extension critical. This must be done with the realization that any validation engine that does not process the extension will reject the certificate (probably limiting the set of applications that can verify the certificate). The a 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 enforce 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 ITU-T Rec. X.660 | ISO/IEC 9834-1. Standard extensions for certificates are defined in clause 8 of this Directory Specification.

The following object class is used to define specific extensions.

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

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

An end-entity certificate is a certificate issued by a CA to a subject that is not an issuer of other public-key certificates.

A CA-certificate is a certificate issued by a CA to a subject that is itself a CA and therefore is capable of issuing public-key certificates. CA-certificates can be themselves categorized by the following types:

- Self-issued certificate – This is a 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 cerificate 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 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.

The directory entry of each user, A, who is participating in strong authentication, contains the certificate(s) of A. Such a certificate is generated by a Certification Authority of A, which is an entity in the DIT. A Certification Authority of A, which may not be unique, is denoted CA(A), or simply CA if A is understood. The public key of A can thus be discovered by any user knowing the public key of CA. Discovering public keys is thus recursive.

If user A, trying to obtain the public key of user B, has already obtained the public key of CA(B), then the process is complete. In order to enable A to obtain the public key of CA(B), the directory entry of each Certification Authority, X, contains a number of certificates. These certificates are of two types. First, there are forward certificates of X generated by other Certification Authorities. Second, there are reverse certificates generated by X itself which are the certified public keys of other certification authorities. The existence of these certificates enables users to construct certification paths from one point to another.

A list of certificates needed to allow a particular user to obtain the public key of another, is known as a *certification path*. Each item in the list is a certificate of the certification authority of the next item in the list. A certification path from A to B (denoted A B):

- starts with a certificate produced by CA(A), namely CA(A)«X1» for some entity X1;
- continues with further certificates Xi«Xi+1»;
- ends with the certificate of B.

A certification path logically forms an unbroken chain of trusted points in the Directory Information Tree between two users wishing to authenticate. The precise method employed by users A and B to obtain certification paths A B and B A may vary. One way to facilitate this is to arrange a hierarchy of CAs, which may or may not coincide with all or part of the DIT hierarchy. The benefit of this is that users who have CAs in the hierarchy may establish a certification path between them using the Directory without any prior information. In order to allow for this each CA may store one certificate and one reverse certificate designated as corresponding to its superior CA.

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

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

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

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.

```
CertPath ::= SEQUENCE OF CrossCertificates

CrossCertificates     ::=     SET OF Certificate
```

Each certificate in a certification path shall be unique. No 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 **CertPath**.

7.1 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 user 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 certification authority’s public key. The use of this card shall additionally be secured by, e.g. at least 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, 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 certification authority 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.

7.2 Public-key certificate creation

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

a) a certification authority shall be satisfied of the identity of a user before creating a certificate for it;

b) a certification authority shall not issue certificates for two users with the same name.

It is important that the transfer of information to the certification authority 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 certificate for a user with a public key that had been tampered with.

b) If the means of generation of key pairs of 7.1 b) or of 7.1 c) is employed, the user's private key must be transferred to the user in a secure manner.

c) If the means of generation of key pairs of 7.1 a) or of 7.1 b) is employed, the user may use different methods (on-line or off-line) to communicate its public key to the CA in a secure manner. On-line 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 transportation to the Directory. As it is produced by an off-line certification authority on behalf of a user who shall be given a copy of it, the user need only store this information in its directory entry on a subsequent access to the Directory. Alternatively the CA could lodge the certificate for the user, in which case this agent shall be given suitable access rights.

7.3 Certificate validity

The authority that issues certificates (public-key or attribute) also has the responsibility to indicate the validity of certificates it issues. Generally, certificates are subject to possible subsequent revocation. This revocation, and 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 authority to perform revocation.

Authorities that do revoke certificates are required to state, through some similar means, what mechanism(s) can be used by relying parties to obtain revocation status information about certificates issued by that authority. This Specification defines a Certificate Revocation List (CRL) mechanism but does not preclude the use of alternative mechanisms. Relying parties check revocation status information, as appropriate, for all certificates considered during path processing procedure described in clause 10 and the delegation path processing procedure described in clause 16 to validate a certificate.

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.

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 of time 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, or the user is no longer to be certified by the authority, or if the authority’s certificate is assumed to be compromised. The revocation of a user certificate or authority certificate shall be made known by the authority, and a new certificate shall be

made available, if appropriate. The authority may then inform the holder of the certificate about its revocation by some off-line 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, on-line certificate status protocol or some other mechanism for the publication of revocation status information;

c) if using CRLs, shall maintain and publish CRLs even if the lists of revoked certificates are empty.

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 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:

– Certificate revocation list;

– Authority revocation list;

– Delta revocation list;

– Attribute certificate revocation list;

– Attribute authority revocation list.

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

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

signature contains the algorithm identifier for the algorithm used by the authority to sign the revocation list.

issuer identifies the entity that has signed and issued the revocation list.

thisUpdate is the date/time at which this revocation list was issued.

nextUpdate, if present, indicates 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 will not be issued any later than the indicated time.

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

crlExtensions, if present, contains one or more CRL extensions.

NOTE 1 — 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 2 — 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 timestamped revocation lists are archived and certified by a current authority.

NOTE 3 — 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 a implementation that only supports version 1 CRLs to still 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.

NOTE 4 — When an implementation processing a certificate revocation list does not recognize a critical extension in the **crlEntryExtensions** field, it shall assume that, at a minimum, the identified certificate has been revoked and is no longer valid and perform additional actions concerning that revoked certificate as dictated by local policy. When an implementation does not recognize a critical extension in the **crlExtensions** field, it shall assume that identified certificates have been revoked and are no longer valid. However in the latter case, since the list may not be complete, certificates that have not been identified as being revoked cannot be assumed to be valid. In this case local policy shall dictate the action to be taken. In any case local policy may dictate actions in addition to and/or stronger than those stated in this Specification.

NOTE 5 — If an extension affects the treatment of the list (e.g. multiple CRLs must be scanned to examine the entire list of revoked certificates, or an entry may represent a range of certificates), then that extension shall be indicated as critical in the **crlExtensions** field regardless of where the extension is placed in the CRL. An extension indicated in the **crlEntryExtensions** field of an entry shall be placed in that entry and shall affect only the certificate(s) specified in that entry.

NOTE 6 — Standard extensions for CRLs are defined in clause 8 of this Directory Specification.

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 certificate user 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 certificate users 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 certificate-using system does not recognize the extension field type or does not implement the semantics of the extension, then that system shall consider the certificate invalid. If an extension is flagged non-critical, a certificate-using system 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 certificate-using system 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 certificate-using implementations which do not need to use such extensions to omit support for them without jeopardizing the ability to interoperate with all certification authorities.

NOTE — A certificate-using system 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 certificate user or may be a CA policy rule indicated to the certificate-using system 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 certificate user, the certification authority 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 can be referenced by a unique identifier, which may be included in the certificate policies extension of the certificate issued by the certification authority, to the end-entity and upon which the certificate user relies. A certificate may be issued in accordance with one or more than one policy. Definition of the policy, and assignment of the identifier, are 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 Specification does not prescribe the style or contents of the certificate policy.

The certificate user may be bound to its obligations under the certificate policy by the act of importing an authority public key and using it as a trust anchor, or by relying on a certificate that includes the associated policy identifier. The certification authority 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 some other means.

For an entity to simply 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 so stated in the policy, certificate users may accept the certification authority'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 Specification.

A certification authority 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 certificate users, 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-certification

A certification authority may be the subject of a certificate issued by another certification authority. In this case, the certificate is called a cross-certificate, the certification authority that is the subject of the certificate is called the subject certification authority and the certification authority that issues the cross-certificate is called an intermediate certification authority (see Figure 2). Both the cross-certificate and the end-entity's certificate may contain a certificate policies extension.

The warranties and obligations shared by the subject certification authority, the intermediate certification authority and the certificate user are defined by the certificate policy identified in the cross-certificate, in accordance with which the subject certification authority may act as, or on behalf of, an end-entity. And the warranties and obligations shared by the certificate subject, the subject certification authority and the intermediate certification authority are defined by the certificate policy identified in the end-entity's certificate, in accordance with which the intermediate certification authority may act as, or on behalf of, a certificate user.

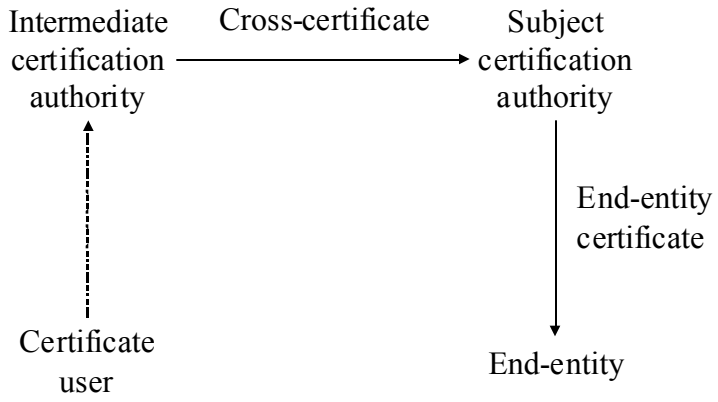


Figure 2 - Cross-certification

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

An intermediate certification authority may, in turn, be the subject of a certificate issued by another certification authority, thereby creating certification paths of length greater than two certificates. And, since trust suffers dilution as certificate paths grow in length, controls are required to ensure that end-entity certificates with an unacceptably low associated trust level will be rejected by the certificate user. This is part of the function of the certification path processing procedure.

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

a) the certification authority does not use the certificate policies extension to convey its policy requirements to certificate users; and

b) the certificate user or intermediate certification authority delegates the job of controlling policy to the next authority in the path.

In the first case, the 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. Certificate users must still ensure that they are using the certificate in conformance with the policies of the authorities in the path.

In the second case, the certificate user or intermediate certification authority should include the special value *any-policy* in the *initial-policy-set* or cross-certificate. Where a 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 certificate user can ensure that all its obligations are conveyed in accordance with the standard by setting the *initial-explicit-policy* indicator. In this way, only authorities that use the standard certificate policies extension as their way of achieving binding are accepted in the path, and certificate users have no additional obligations. Because authorities also attract obligations when they act as, or on behalf of, a certificate user, they can ensure that all their obligations are conveyed in accordance with the standard by setting **requireExplicitPolicy** 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 certification authority issues certificates to end-entities, even though the policy authorities under which the two certification authorities operate may have selected different unique identifiers for these materially equivalent policies. In this case, the intermediate certification authority may include a policy mappings extension in the cross-certificate. In the policy mappings extension, the intermediate certification authority assures the certificate user that it will continue to enjoy the familiar warranties, and that it should continue to fulfill its familiar obligations, even though subsequent entities in the certification path operate in a different policy domain. The intermediate certification authority 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 certification authority operates is identical to those according to which the intermediate certification authority 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 certificate user.

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

Certificate users may determine that certificates issued in a policy domain other than its own should not be relied upon, even though a trusted intermediate certification authority 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 certification authority may make a similar determination on behalf of its certificate users. In order to ensure that certificate users correctly enforce this requirement, it can set **inhibitPolicyMapping** in a policy constraints extension.

8.1.4 Certification path processing

The certificate user 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 certificate user knows, a priori, the set of policies that are acceptable for its intended use.

The second strategy may be most appropriate when the certificate user 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 certificate user must 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 certificate user can cause the procedure to return a valid result if the path is valid under any (unspecified) policy.

8.1.5 Self-issued certificates

There are three circumstances under which a certification authority may issue a certificate to itself:

a) as a convenient way of encoding its public key for communication to, and storage by, its certificate users;

b) for certifying key usages other than certificate and CRL signing (such as time-stamping); and

c) for replacing its own expired certificates.

These types of certificate 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 purposes of path validation, self-issued certificates of type a) are 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 certification authority 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 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 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.

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 certificate-using system 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 certificate user 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 certificate-using system 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 signature verifier 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 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 noted otherwise, 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 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 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, which indicates the purpose for which the certified public key is used, is defined as follows:

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

KeyUsage ::= BIT STRING {
      digitalSignature        (0),
      nonRepudiation          (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 have purposes other than those identified in b), f), or g) below;

b) **nonRepudiation**: for verifying digital signatures used in providing a nonrepudiation service which protects against the signing entity falsely denying some action (excluding certificate or CRL signing, as in f) or g) below);

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;

g) **cRLSign**: for verifying a CA's signature on CRLs.

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);

The bit **keyCertSign** is for use in CA-certificates only. If **KeyUsage** is set to **keyCertSign** and the basic constraints extension is present in the same certificate, the value of the **cA** component of that extension shall be set to **TRUE**. CAs may also use other of the defined key usage bits in **KeyUsage**, e.g. **digitalSignature** for providing authentication and integrity of on-line administration transactions.

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

If the extension is flagged critical, then the certificate shall be used only for a purpose for which the corresponding key usage bit is set to one.

If the extension if 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 certificate-using system recognizes and processes the **keyUsage** extension type, then the certificate-using system shall ensure that the certificate shall be used only for a purpose for which the corresponding key usage bit is set to one. A bit set to zero indicates that the key is not intended for that purpose. If all bits are zero, it indicates the key is intended for some purpose other than those listed.

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
```

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

This extension may, at the option of the certificate issuer, 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 certificate-using system recognizes and processes the **extendedKeyUsage** extension type, then the certificate-using system 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.

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 key has not been revoked, for example due to key compromise, up to the time of verification, then a valid certificate must 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 certificate-using system 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 certificate user 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 certificate user, 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 CCITT Rec.

X.660 | ISO/IEC 9834-1. A CA may assert any-policy by using the **anyPolicy** 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 Specification. No object identifiers will be assigned in this Specification for specific certificate policies. That assignment is the responsibility of the entity that defines the certificate policy.

```
anyPolicy      OBJECT IDENTIFIER ::=      { 2 5 29 32 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 certificate user 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 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, X.400 originator/recipient addresses, and EDI party names. It is therefore necessary to be able to securely associate multiple names of a variety of name forms with a certificate subject or a certificate or CRL issuer.

b) A certificate user may need to securely 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 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 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]   DirectoryString {ub-name} OPTIONAL,
  partyName       [1]   DirectoryString {ub-name} }
```

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

— **otherName** is a name of any form defined as an instance of the **OTHER-NAME** information object class;

— **rfc822Name** is an Internet electronic mail address defined in accordance with Internet RFC 822;

— **dNSName** is an Internet domain name defined in accordance with Internet RFC 1035;

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

— **directoryName** is a directory name defined in accordance with ITU-T Rec. X.501 | ISO/IEC 9594-2;

— **ediPartyName** 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;

— **uniformResourceIdentifier** is a Uniform Resource Identifier for the World-Wide Web defined in accordance with Internet RFC 1630;

— **iPAddress** is an Internet Protocol address defined in accordance with Internet RFC 791, represented as a binary string.

— **registeredID** is an identifier of any registered object assigned in accordance with CCITT Rec. 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 certificate issuer and certificate users.

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 certificate-using system is permitted to ignore any name with an unrecognized or unsupported name form. It is recommended that, provided the subject field of the certificate contains a directory 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 ITU-T Rec. X.681 | ISO/IEC 8824-2.

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

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 is present must be recognized and processed, otherwise the certificate or CRL shall be considered invalid. Apart from the preceding restriction, a certificate-using system 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 directory name that unambiguously identifies the issuing authority, this field be flagged non-critical.

NOTE — If this extension field is present and is flagged critical, the **issuer** field of the certificate or CRL may contain a null name (e.g. a sequence of zero relative distinguished names) in which case the issuer is identified only by the name or names in this extension.

8.3.2.3 Subject directory attributes extension

This field conveys any desired Directory attribute values for 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
```

This extension is always non-critical.

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 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 certificate user 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 certificate user 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 certificate user.

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 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. Directory name structures considered natural for organizations or geographical areas shall not need adjustment in order to accommodate certification authority requirements.

i) Certificate extension fields need to be backward-compatible with the unconstrained certification path approach system as specified in earlier editions of ITU-T Rec. X.509 | ISO/IEC 9594-8.

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

NOTE — In any certificate-using system, 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 must 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) Basic constraints;

b) Name constraints;

c) Policy constraints;

d) Inhibit any policy.

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 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 indicates if the certified public key may be used to verify certificate signatures.

The **pathLenConstraint** component shall be present only if **cA** is set to true. It gives the maximum number of CA-certificates that may follow this certificate in a certification path. Value 0 indicates that the subject of this certificate may issue certificates only to end-entities and not to further CAs. If no **pathLenConstraint** field appears in any certificate of a certification path, there is no limit to the allowed length of the certification path.

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 an entity which is not authorized to be a CA may issue certificates and a certificate-using system 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 certificate-using system, 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 certificate-using system shall check that the certification path being processed is consistent with the value of **pathLenConstraint**.

NOTE 1 — If this extension is not present, or is flagged non-critical and is not recognized by a certificate-using system, then the certificate is to be considered an end-entity certificate and cannot be used to verify certificate signatures.

NOTE 2 — To constrain a certificate subject to being only an end entity, i.e. not a CA, the issuer can 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 a name space within which all subject names in subsequent certificates in a certification path must be located. 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 }

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

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

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

If present, the **permittedSubtrees** and **excludedSubtrees** components each specify one or more naming subtrees, each defined by the name of the root of the subtree and, optionally, within that subtree, an area that is bounded by upper and/or lower levels. If **permittedSubtrees** is present, of all the certificates issued by the subject CA and subsequent CAs in the certification path, only those certificates with subject names within these subtrees are 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 and the name spaces overlap, the exclusion statement takes precedence.

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. Conformant implementations are not required to recognize all possible name forms. If the extension is flagged 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 non-critical and a certificate-using implementation does not recognize a name form used in any **base** component, then that subtree specification may be ignored. When a certificate subject has multiple names of the same name form (including, in the case of the **directoryName** name form, the name in the subject field of the certificate if non-null) then all such names shall be tested for consistency with a name constraint of that name form.

NOTE — When testing certificate subject names for consistency with a name constraint, names in non-critical subject alternative name extensions should be processed, not ignored.

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 naming 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 naming subtree excludes all nodes except the subtree base and its immediate subordinates.

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 certificate user may not check that subsequent certificates in a certification path are located in the name space intended by the issuing CA.

If this extension is present and is flagged critical, or is flagged non-critical but is recognized by the certificate-using system, then the certificate-using system shall check that the certification path being processed is consistent with the value in this 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 }

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

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 subject 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 certificate user 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 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 certificate user 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) Certificate users need to be able to track all CRLs issued from a CRL issuer or CRL distribution point (see 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 occured, and this information may be valuable to a certificate user. 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.

e) Certificate users 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.

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

g) Delta CRLs may also be available that update a given base CRL. Certificate users 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.

8.5.2 CRL and CRL entry extension fields

The following extension fields are defined:

a) CRL number;
b) Reason code;
c) Hold instruction code;
d) Invalidity date;
e) CRL scope;
f) Status referral;
g) CRL stream identifier;
h) Ordered list;
i) Delta information.

The CRL number, CRL scope, status referral, CRL stream identifier, ordered list and delta information shall be used only as a CRL extension field and the other fields shall be used only as CRL entry extension fields.

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 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:

- **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 certificate users (see 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; or,

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; or,

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

The **removeFromCRL** reason code is for use with delta-CRLs (see 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.

8.5.2.3 Hold instruction code extension

This CRL entry extension field provides for 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.”

8.5.2.4 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 nonrepudiation purposes. For example, this date may be a date advised by the private key holder, and it is possible for such a person to fraudulently claim that a key was compromised some time 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.

8.5.2.5 CRL scope extension

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, must 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 which type(s) of certificates the CRL contains revocation status information concerning. 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 bit0 of the **BIT STRING** is set, then an additional zero octet should be prepended so as to ensure the resulting encoding represents a positive **INTEGER**. e.g.:

03 02 01 f7 (represents bits 0-6 set)

maps to

02 02 00 f7 (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 is 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 must 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 certificate falls within the scope of the CRL if and only if it meets the criteria of both extensions. If the CRL contains neither an **issuingDistributionPoint** nor **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 certificate-using system 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 certificate-using system must 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 certificate-using system must check that the reason codes covered by this CRL are adequate for 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 the only those reason codes included in the **onlySomeReasons** components of both extensions are covered by this CRL.

8.5.2.6 Status referral extension

This CRL extension is for use within the CRL structure as a means to convey information about revocation notices to certificate users. 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 certificate users or relying parties as a source of revocation notices, but rather as a tool to ensure that the appropriate revocation information is used.

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 or not 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 certificate user, on obtaining this list, can quickly determine if cached copies of CRLs are still up-to-date. This may eliminate much unnecessary retrieval of CRLs. Furthermore, by using this mechanism, certificate users become aware of CRLs issued by the authority between its usual update cycle, 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 certificate users. 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 they are 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** field identifies the entity that signs the CRL; this defaults to the issuer name of the encompassing CRL.

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

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

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

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

The **lastChangedCRL** 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 Specification, to locate appropriate revocation status information.

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

8.5.2.7 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, must 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.

8.5.2.8 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.

8.5.2.9 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.

8.6 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 must 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 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) Certificate issuer;

d) Delta CRL indicator;

e) Base update;

f) Freshest CRL.

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

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 certificate user should refer to ascertain if the certificate has been revoked. A certificate user 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** field of the issuing distribution point extension of the CRL. A certificate-using system 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 certificate-using system 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 directory name that is directly subordinate to the directory name of the CRL issuer. In this case, the **nameRelativeToCRLIssuer** component conveys the relative distinguished name with respect to the CRL issuer directory 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 certificate-using system 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 certificate-using system 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 subclause 8.5.2.2 of this Specification.

8.6.2.2 Issuing distribution point extension

This CRL extension field identifies the CRL distribution point for this particular CRL, and indicates if the CRL is limited to revocations for end-entity certificates only, for authority certificates only, or for a limited set of reasons only. The CRL is signed by the CRL issuer's 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 this field is absent, the CRL shall contain entries for all revoked unexpired certificates issued by the CRL issuer.

This field is defined as follows:

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

IssuingDistPointSyntax ::= SEQUENCE {

  distributionPoint              [0] DistributionPointName OPTIONAL,
  onlyContainsUserCerts          [1] BOOLEAN DEFAULT FALSE,
  onlyContainsAuthorityCerts     [2] BOOLEAN DEFAULT FALSE,
  onlySomeReasons                [3] ReasonFlags OPTIONAL,
  indirectCRL                    [4] BOOLEAN DEFAULT FALSE,
  onlyContainsAttributeCerts     [5] BOOLEAN DEFAULT FALSE }
```

The **distributionPoint** component contains the name of the distribution point in one or more name forms. If this field is absent, the CRL shall contain entries for all revoked certificates issued by the CRL issuer. After a certificate appears on a CRL, it may be deleted from a subsequent CRL after the certificate's expiry.

If **onlyContainsUserCerts** is true, the CRL only contains revocations for end-entity certificates. If **onlyContainsAuthorityCerts** is true, the CRL only contains revocations for authority certificates. If **onlySomeReasons** is present, the CRL only contains revocations 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 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 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 **onlyContainsUserCerts**, **onlyContainsAuthorityCerts**, and **onlySomeReasons** indicators, from all authorities that identify this CRL issuer in their certificates.

If **onlyContainsAttributeCerts** is TRUE, the CRL only contains revocations for attribute certificates.

For CRLs distributed via the Directory, the following rules regarding use of attributes apply. A CRL which has **onlyContainsAuthorityCerts** set 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. Otherwise the CRL 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 authority entry.

This extension is always critical. A certificate user which 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 must contain all current CRL entries for the issuing authority, including entries for all revoked user certificates and authority certificates.

NOTE 1 — The means by which revocation information is communicated by authorities to CRL issuers is beyond the scope of this Recommendation | International Standard.

NOTE 2 — If a authority issues, from its own directory entry (i.e. not from a separately-named CRL distribution point), a CRL with **onlyContainsUserCerts** or **onlyContainsAuthorityCerts** set, then the authority should ensure that all 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 ignored this extension it could not correctly attribute CRL entries to certificates.

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 certificate user 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 shall be used only as a certificate extension and may be used in certificates issued to authorities as well as certificates issued to users. This field identifies the CRL to which a certificate user 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 }
```

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 certificate-using system 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 or not the freshest CRL is required to be checked.

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 certificate user 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** component for each base CRL for which the indirect dCRL provides updated information.

Application of a dCRL to the referenced base revocation information must 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.
    - 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 must 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** component of the **deltaCRLIndicator** extension or the **cRLNumber** element of the **BaseRevocationInfo** component of the **cRLScope** extension. The certificate must 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 must be listed with the revocation reason appropriate for the subsequent revocation.
    - If the certificate was not removed from hold, but was permanently revoked, then it must 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** component of the **deltaCRLIndicator** extension or the **cRLNumber** element of the **BaseRevocationInfo** component of the **cRLScope** extension.

- A certificate's revocation notice may first appear on a 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 must 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 CRLreferenced 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.

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;

Note: Each certificate in a certification path is unique. A path that contains the same certificate two or more times is not a valid 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 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 certificate user for the purposes of certification path processing; this input can also take the special value *any-policy*;

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 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-any-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; and

g) the current date/time (if not available internally to the certification path processing module).

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 certificate user 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 certificate user 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 certificate user 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 certificate-using system 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 must 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) *explicit-policy-indicator*: Indicates whether an acceptable policy must be explicitly identified in every certificate in the path;

e) *path depth*: An integer equal to one more than the number of certificates in the certification path for which processing has been completed;

f) *policy-mapping-inhibit-indicator*: Indicates whether policy mapping is inhibited;

g) inhibit-*any-policy-indicator:* Indicates whether the special value **anyPolicy** is considered a match for any specific certificate policy;

h) *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:

a) Write *any-policy* in the zeroth and first columns of the zeroth row of the *authorities-constrained-policy-set* table;

b) Initialize the *permitted-subtrees* variable to *unbounded*;

c) Initialize the *excluded-subtrees* variable to an empty set;

d) Initialize the *explicit-policy-indicator* to the *initial-explicit-policy* value;

e) Initialize *path-depth* to one;

f) Initialize the *policy-mapping-inhibit-indicator* to the *initial-policy-mapping-inhibit* value;

g) Initialize the inhibit-any-policy-indicator to the initial-inhibit-any-policy value;

h) 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:

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 certificate, if the basic constraints extension field is present in the certificate, check that the **cA** component is present and set to 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, 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*.

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:

a) 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.

b) 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.

c) If policy-mapping-inhibit-indicator is set:

— process any policy mappings extension by, for each mapping identified in the extension, locate 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.

d) If *policy-mapping-inhibit-indicator* is not set:

— process any policy mappings extension by, for each mapping identified in the extension, locate 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 must be 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** constraint 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).

e) For any row not modified in either step c) or d), 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.

f) 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** constraint 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).

g) Increment *path-depth*.

10.5.3 Explicit policy indicator processing

For all certificates, the following actions are then performed:

a) 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** constraint 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 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

For the end-certificate, the following actions are then performed:

a) If *explicit-policy-indicator* is set, check that the *authorities-constrained-policy-set* table is not empty. If any of the above checks were to fail, then the procedure shall terminate, returning a failure indication, an appropriate reason code, *explicit-policy-indicator* and null values in the *user-constrained-policy-set* and the *authorities-constrained-policy-set* table.

If none of the above checks were to fail on the end certificate, then the *user-constrained-policy-set* shall be calculated by forming the intersection of the *authorities-constrained-policy-set* and the *initial-policy-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*. Then the procedure shall terminate, returning a success indication together with the *explicit-policy-indicator*, the *authorities-constrained-policy-set* table and the *user-constrained-policy-set.* If the intersection of the authority constrained set and user constrained set is null, the path is valid under authority constrained policy(s), but none is acceptable to the user.

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 & 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 certificate 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 of structural object class **pkiCA**.

```
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 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** elements 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 must 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, 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
                                 -- at least one of the pair shall be present -- }
  (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,
  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 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   DirectoryString {ub-content},
  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.

If **content** is present, the complete content of the certificate policy is included.

If **pointer** is present, 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.

11.2.10 PKI path attribute

The PKI path attribute is used to store certification paths, each consisting of a sequence of cross-certificates.

```
pkiPathATTRIBUTE  ::= {
  WITH SYNTAX      PkiPath
   ID              id-at-pkiPath }

PkiPath ::= SEQUENCE OF CrossCertificates
```

This attribute can be stored in the CA directory entry and would contain some certification paths from that CA to other CAs. This attribute, if used, enables more efficient retrieval of cross-certificates that form frequently used certification 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 will likely not represent the complete set of forward certification paths for any given CA.

11.3 PKI directory matching rules

This Directory Specification defines matching rules for use with attributes of type **Certificate**, **CertificatePair**, **CertificateList, CertificatePolicy**, and **SupportedAlgorithm**, respectively. This clause also defines matching rules to facilitate the selection of certificates or CRLs with specific characteristics from multi-valued attributes holding multiple certificates or CRLs.

11.3.1 Certificate exact match

The certificate exact match rule compares for equality a presented value with an attribute value of type **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 of type **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 must 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 on the basis of 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
  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 CA that issued the end-entity certificate being validated.

```
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.

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 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 revocation of attribute certificates is not required. Privilege delegation and the use of roles are also aspects of this framework that are not universally applicable. However these are included in this Specification so that those environments that do have requirements for them can also be supported.

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 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 of this Specification 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 assignment of privilege will frequently be 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 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 Certification Authority (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 a certificate are outside the scope of this Specification.

The attribute certificate is defined as follows.

```
AttributeCertificate ::= SIGNED {AttributeCertificateInfo}

AttributeCertificateInfo ::= SEQUENCE
  {
  version                 AttCertVersion, --version is v2
  holder                  Holder,
  issuer                  AttCertIssuer,
  signature               AlgorithmIdentifier,
  serialNumber            CertificateSerialNumber,
  attrCertValidityPeriod  AttCertValidityPeriod,
  attributes              SEQUENCE OF Attribute,
  issuerUniqueID          UniqueIdentifier OPTIONAL,
  extensions              Extensions      OPTIONAL
  }


AttCertVersion        ::= INTEGER { v2(1) }

Holder  ::=     SEQUENCE
          {
          baseCertificateID     [0] IssuerSerial                OPTIONAL,
                -- the issuer and serial number of  the holder's Public Key Certificate
          entityName            [1] GeneralNames                OPTIONAL,
                -- the name of the entity or role
          objectDigestInfo      [2] ObjectDigestInfo            OPTIONAL
                used to directly authenticate the holder, e.g. an executable
  --at least one of baseCertificateID, entityName or objectDigestInfo must be present--}

ObjectDigestInfo    ::= SEQUENCE {
  digestedObjectType  ENUMERATED {
          publicKey             (0),
          publicKeyCert         (1),
          otherObjectTypes      (2) },
  otherObjectTypeID             OBJECT IDENTIFIER  OPTIONAL,
  digestAlgorithm               AlgorithmIdentifier,
  objectDigest                  BIT STRING }


   AttCertIssuer ::=    [0]      SEQUENCE {
  issuerName                    GeneralNames  OPTIONAL,
  baseCertificateID     [0]     IssuerSerial  OPTIONAL,
  objectDigestInfo      [1]     ObjectDigestInfo  OPTIONAL }
-- At least one component must be present
  ( WITH COMPONENTS { ..., issuerName  PRESENT } |
    WITH COMPONENTS { ..., baseCertificateID  PRESENT } |
    WITH COMPONENTS { ..., objectDigestInfo PRESENT } )
```

```
IssuerSerial  ::=  SEQUENCE {
  issuer         GeneralNames,
  serial         CertificateSerialNumber,
  issuerUID      UniqueIdentifier OPTIONAL }

AttCertValidityPeriod  ::= SEQUENCE {
  notBeforeTime      GeneralizedTime,
  notAfterTime       GeneralizedTime }
```

The **version** differentiates between different versions of the attribute certificate. For attribute certificates issued in accordance with the syntax in this Specification, **version** must be **v2**.

The **holder** field conveys the identity of the attribute certificate's holder.

The **baseCertificateID** component, if present, identifies a particular public-key certificate that is to be used to authenticate the identity of this holder when asserting privileges with this attribute certificate.

The **entityName** component, if present, identifies 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 **baseCertificateID** and **entityName** are both present, only the certificate specified by **baseCertificateID** may be used. In this case **entityName** is included only as a tool to help the privilege verifier locate the identified public-key certificate.

NOTE — 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 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.

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 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 must be 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**. 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) then 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 conveys the identity of the AA that issued the certificate.

— The **issuerName** component, if present, identifies one or more names for the issuer.

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

— The **objectDigestInfo** component, if present, identifies the issuer by providing a hash of identifying information for the issuer.

The **signature** identifies the cryptographic algorithm used to digitally sign the attribute certificate.

The **serialNumber** is the serial number that uniquely identifies the attribute certificate within the scope of its issuer.

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

The **attributes** field contains the attributes associated with the holder that are being certified (e.g. the privileges).

NOTE — In the case of attribute descriptor attribute certificates, this sequence of attributes can be empty.

The **issuerUniqueID** may be used to identify the issuer of the attribute certificate in instances where the issuer component is not sufficient.

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

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 certificate paths

Just as with public-key certificates, there may be a requirement to convey an attribute certificate path (e.g. within an application protocol to assert privileges). The following ASN.1 data type can be used to represent an attribute certificate 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 (e.g. by including such a designation in their identity certificates).

The Source of Authority (SOA) is the entity that is trusted by a privilege verifier as the entity with ultimate responsibility for assignment of a set a 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 certificates to other entities in which privileges are assigned to those entities. An SOA is analogous to a ‘root CA’ or ‘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 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 some other means (e.g. locally configured).

c) 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.

d) 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 **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.

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.

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 some 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 some AA. Once created, this certificate may be returned (only) to the requesting entity, which explicitly supplies it when requesting access to some protected resource.

Note that in both procedures the AA must 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 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 re-using 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 some 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. An example is provided in Annex D.

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 must 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 Specification.

Privilege policy specifies the threshold for acceptance for a given set of privileges. That is, it defines precisely when a privilege verifier should conclude that a presented set of privileges is "sufficient" in order that it may grant access (to the requested object, resource, application, etc.) to the privilege asserter.

Syntax for the definition of privilege policy is not standardized in this Specification. Annex D 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 must 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 some 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 PMI in the 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 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 (ITU-T Rec. X.812 | ISO/IEC 10181-3) that defines a corresponding set of terms that are specific to the access control application. A mapping of the generic terms used in this Specification to those in the access control framework is provided here, to clarify the relationship between this model and that specification.

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

Privilege verifier in this Specification would be acting in the role of an ‘access control decision function (ADF)’ in the access control framework.

Object method for which privilege is being asserted in this Specification would correspond to the ‘target’ defined in the access control framework.

Environmental variables in this Specification would correspond to the ‘contextual information’ in the access control framework.

Privilege policy discussed in this Specification could include ‘access control policy’, and ‘access control policy rules’ as defined in the access control framework.

This model allows a PMI to be overlaid fairly seamlessly on an existing network of resources to be protected. In particular, having the privilege verifier act as a gateway to a sensitive object method, granting or denying requests for invocation of that object method, enables the object to be protected with little or no impact to the object itself. The privilege verifier screens all requests and only those that are properly authorized are passed on to the appropriate object methods.

14.1.2 PMI in a non-repudiation context

There is a standard framework for non-repudiation (ITU-T Rec. X.813 | ISO/IEC 10181-4) which defines a corresponding set of terms that are specific to non-repudiation. A mapping of the generic terms used in this Specification to those in the non-repudiation framework is provided here, to clarify the relationship between this model and that specification.

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

Privilege verifier in this Specification would be acting in the role of an ‘evidence user’ or a ‘recipient’ in the non-repudiation framework.

Object method for which privilege is being asserted in this Specification would correspond to the 'target' defined in the non-repudiation framework.

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

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

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 3). 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.

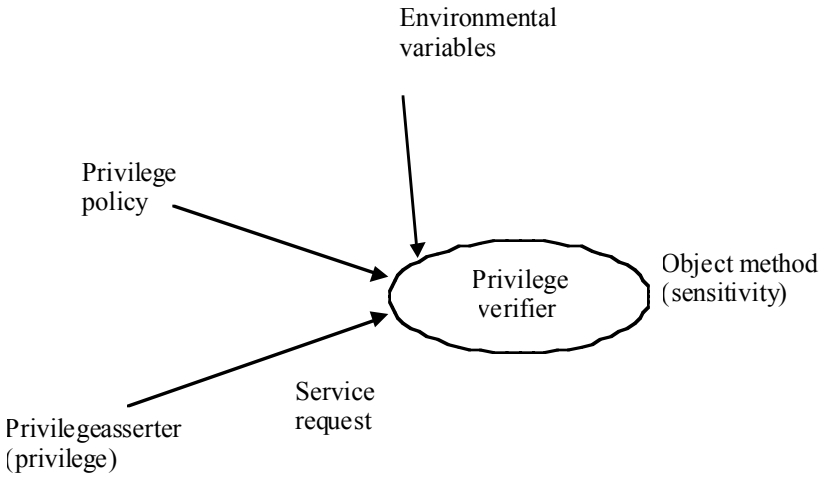


Figure 3 - Control model

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 4).

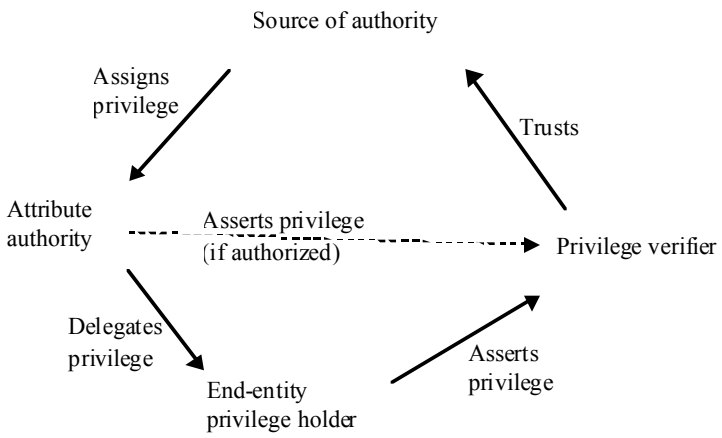


Figure 4 - Delegation model

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 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 must locate a delegation path of certificates from that of the privilege asserter to one issued by the SOA. The validation of that delegation path must include 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 must 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. Ony AAs may delegate privilege. End-entities cannot.

14.4 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 some 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 must 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 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 must 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 must 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.

14.4.1 Role attribute

The specification of privilege attribute types is generally an application-specific issue that is outside the scope of this Specification. The single exception to this is an attribute defined here for the assignment of a holder to a role. The specification of values for the role attribute is outside the scope of this Specification.

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

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

This privilege attribute would be used to populate the **attributes** field of a role assignment certificate. If the role assignment certificate is a public-key certificate, this attribute would be used to populate the **subjectDirectoryAttributes** extension of that public-key certificate.

The **roleAuthority**, if present, identifies the recognized authority that is responsible for issuing the role specification certificate.

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** must be present in the **issuer** field of that role specification certificate. If the privilege verifier 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 Specification.

If **roleAuthority** is absent, the identity of the responsible authority must 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 a role assignment certificate containing this attribute is assigned. If a privilege verifier uses a role specification certificate to determine the privileges assigned to that role, this role name must also appear in the **holder** field of the role specification certificate.

15 Privilege management certificate extensions

The following certificate extensions may be included in certificates for 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 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 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 location of related role specification certificates.

e) *Delegation*: These certificate extensions allow constraints to be set on subsequent delegation of assigned privileges.

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.

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.

15.1.2.1 Time specification extension

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 am and 5:00 pm. Another example, in the case of delegation, might be a manager delegating 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 must 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 must be rejected.

15.1.2.1.1 Time specification match

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 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 must 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 must 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 must 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 must be rejected.

15.2 Privilege revocation extensions

15.2.1 Requirements

The following requirements relate to 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 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

The SOA identifier extension indicates that the certificate subject may act as an SOA for 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 mechansims 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 must 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.

This extension is always non-critical.

15.3.2.2 Attribute descriptor extension

The definition of a privilege attribute, and the domination rules governing subsequent delegation of that privilege, are needed by privilege verifiers to ensure that authorization is done correctly. These definitions and rules may be provided to privilege verifiers in a variety of ways outside the scope of this Specification (e.g. they may be locally configured at the privilege verifier).

This extension provides one mechanism that can be used by an SOA to make privilege attribute definitions and associated domination rules available to privilege verifiers. An attribute certificate that contains this extension is called an attribute descriptor certificate and is a special type of attribute certificate. Although syntactically identical to an **AttributeCertificate** an attribute descriptor certificate:

- contains an empty **SEQUENCE** in its **attributes** 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 ITU-T Rec. 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.

15.3.2.2.1 Attribute descriptor match

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 requirements relate to roles:

a) 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:

a) Role specification certificate identifier.

15.4.2.1 Role specification certificate identifier extension

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 must 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.

15.4.2.1.1 Role specification certificate ID match

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 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 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 must 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.

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.

15.5.2.1 Basic attribute constraints extension

This field indicates whether 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. Therefore the total length of the path may exceed the value of the constraint by as many as two certificates. This includes the certificates at the two endpoints plus the AA certificates between the two endpoints which 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 which 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.

15.5.2.1.1 Basic attribute constraints match

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

The delegated name constraints field indicates a name space within which all holder names in subsequent certificates in a delegation path must 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 certificate user may not check that subsequent attribute certificates in a delegation path are located in the name space intended by the issuing AA.

15.5.2.2.1 Delegated name constraints match

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

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 must 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 must 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 must 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 must be rejected.

15.5.2.3.1 Acceptable certificate policies match

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

In privilege delegation, an AA that delegates privileges, must 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 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.

15.5.2.4.1 AA identifier match

The authority attribute identifier matching rule compares for equality a presented value with an attribute value of type **AttributeCertificate.**

```
authAttIdMatch MATCHING-RULE ::= {
    SYNTAX      AuthorityAttirbuteIdentifierSyntax
    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.

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 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 16.2 below. If the privilege asserter was delegated its privilege by an intermediary AA, then the delegation path procedure in 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 must be verified. Procedures related to validating signatures and public-key certificates are not repeated in this clause. The privilege verifier must 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. 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 must determine if the privilege holder actually intended to assert the contained privilege for use with that context. The fact that a chain of certificates to a trusted SOA exists is not in itself enough upon which to make this determination. The willingness of the privilege holder to use that certificate has to be clearly indicated and verified. However, mechanisms to ensure that such a privilege assertion has been adequately demonstrated by the privilege holder are outside the scope of this Specification. As an example, such a privilege assertion may be verifiable if the privilege holder signed a reference to that certificate, thereby indicating their willingness to use that certificate for that context.

For each attribute certificate in the path that does not contain the **noRevAvail** extension, the privilege verifier must ensure that the attribute certificate has not been revoked.

The privilege verifier must 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 must 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 must 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 must check that the server/service for which it is verifying is included in the list of targets.

If the certificate is a role assignment certificate, the processing procedure described in 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 16.3 is needed to ensure that delegation was done properly.

The privilege verifier must 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 specification of any environmental variables that need to be considered. The privileges asserted, including those resulting from the role procedure in 16.2 and the delegation procedure in 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 must 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 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 must 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.

Prior to commencing delegation path validation, the privilege verifier must obtain the following. Any of these may be provided by the privilege asserter, or obtained by the privilege verifier from some other source, such as the Directory. The attributes of the service may be provided to the privilege verifier in a structured document or by some 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.

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 must 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 must 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 locate the appropriate certificate of the adjacent entity in the delegation path. 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 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.

16.3.3 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 must 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 must 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 16.2. The comparison of the two privileges is done based on the domination rule discussed in 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 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 section 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 ::= {
-- a PMI user (i.e., a "holder")
  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 object class

The PMI delegation path object class is used in defining entries for objects that may contain delegation paths. It will generally be used in conjunction with entries of structural object class **pmiAA**.

```
pmiDelegationPath   OBJECT-CLASS      ::= {
      SUBCLASS OF       {top}
```

```
        KIND                auxiliary
        MAY CONTAIN         { delegationPath }
        ID                  id-oc-pmiDelegationPath }
```

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 }
```

17.2 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 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.

17.3 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  OPTIONAL,
issuer              IssuerSerial }
```

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 must 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.

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 authentication of users accessing the Directory via DUAs and 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. 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 signaling of procedures for protecting passwords may be a matter for 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 5.

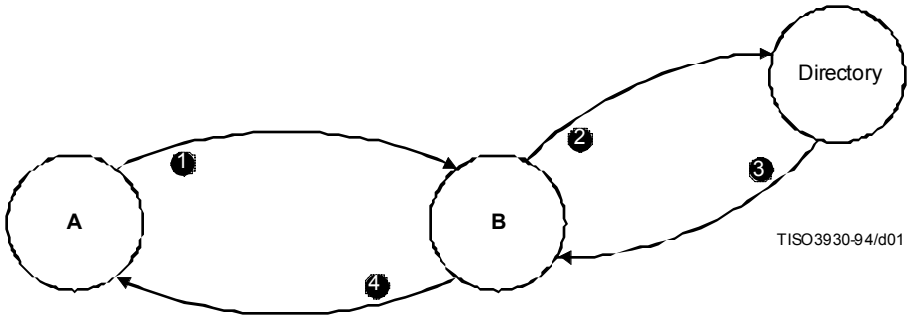


Figure 5 – The unprotected simple authentication procedure

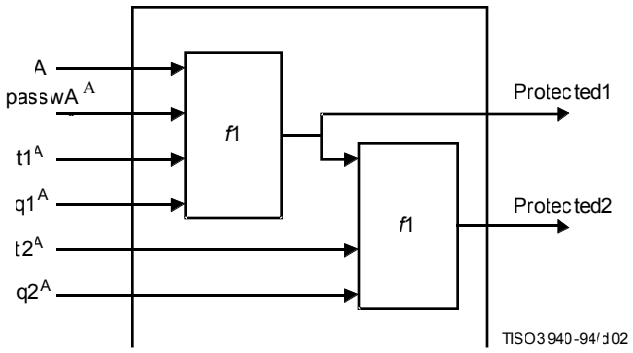
The following steps are involved:

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 6 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.



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

Figure 6 – Protected simple authentication

18.1.2 Procedure for protected simple authentication

Figure 7 illustrates the procedure for protected simple authentication.

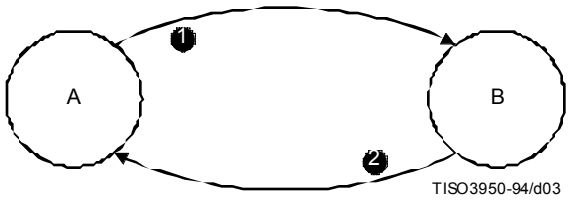


Figure 7 – The protected simple authentication procedure

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 (*f*1) of Figure 6, 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:

   $Protected1 = f1\ (t1^A, q1^A, A, passwA^A)$

   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 6. The further protection is of the form:

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

   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 ITU-T Rec. X.511 | ISO/IEC 9594-3 and ITU-T Rec. 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

A User Password attribute type contains the password of an object. An attribute value for the user password is a string specified by the object.

```
userPassword    ATTRIBUTE   ::=     {
        WITH SYNTAX                 OCTET STRING (SIZE (0..ub-user-password))
        EQUALITY MATCHING RULE      octetStringMatch
        ID                          id-at-userPassword }
```

18.2 Strong Authentication

The procedures described in this subclause are for use in authentication between a DUA and a DSA as well as between pairs of DSAs. The procedures make use of the public-key certificate framework defined in this Specification. In addition, the procedures make use of the Directory itself as the repository for public-key information required to perform the authentication. The inclusion of relevant parameters in Directory protocols is defined in the protocol specifications themselves. The procedures defined here for strong authentication may also be used by applications other than the Directory that also make use of such a repository. For the Directory use of these procedures the term 'user' in these procedures can refer to either a DUA or a DSA.

The approach to strong authentication taken in this Directory Specification makes use of the properties of a family of cryptographic systems, known as public-key cryptosystems (PKCS). These cryptosystems, also described as asymmetric, involve a pair of keys, one private and one public, rather than a single key as in conventional cryptographic systems. Annex E gives a brief introduction to these cryptosystems and the properties which make them useful in authentication. For a PKCS to be usable in this authentication framework at this present time, it shall have the property that both keys in the key pair can be used for encipherment, with the private key being used to decipher if the public key was used, and the public key being used to decipher if the private key was used. In other words, $X_p \bullet X_s = X_s \bullet X_p$, where X_p/X_s are encipherment/decipherment functions using the public/private keys of user X.

NOTE — Alternative types of PKCS, i.e. ones which do not require the property of permutability and that can be supported without great modification to this Directory Specification, are a possible future extension.

This authentication framework does not mandate a particular cryptosystem for use. It is intended that the framework shall be applicable to any suitable public key cryptosystem, and shall thus support changes to the methods used as a result of future advances in cryptography, mathematical techniques or computational capabilities. However, two users wishing to authenticate shall support the same cryptographic algorithm for authentication to be performed correctly. Thus, within the context of a set of related applications, the choice of a single algorithm shall serve to maximize the community of users able to authenticate and communicate securely.

Authentication relies on each user possessing a unique distinguished name. The allocation of distinguished names is the responsibility of the Naming Authorities. Each user shall therefore trust the Naming Authorities not to issue duplicate distinguished names.

Each user is identified by its possession of its private key. A second user is able to determine if a communication partner is in possession of the private key, and can use this to corroborate that the communication partner is in fact the user. The validity of this corroboration depends on the private key remaining confidential to the user.

For a user to determine that a communication partner is in possession of another user's private key, it shall itself be in possession of that user's public key. Whilst obtaining the value of this public key from the user's entry in the Directory is straightforward, verifying its correctness is more problematic. There are many possible ways for doing this: subclause 18.2.1 describes a process whereby a user's public key can be checked by reference to the Directory. This process can only operate if there is an unbroken chain of trusted points in the Directory between the users requiring to authenticate. Such a chain can be constructed by identifying a common point of trust. This common point of trust shall be linked to each user by an unbroken chain of trusted points.

18.2.1 Obtaining public-key certificates from the directory

Certificates are held within directory entries as attributes of type **UserCertificate**, **CACertificate** and **CrossCertificatePair**. These attribute types are known to the Directory. These attributes can be operated on using the same protocol operations as other attributes. The definition of these types can be found in 3.3; the specification of these attribute types is defined in subclause 11.2.

In the general case, before users can mutually authenticate, the Directory shall supply the complete certification and return certification paths. However, in practice, the amount of information which shall be obtained from the Directory can be reduced for a particular instance of authentication by:

a) if the two users that want to authenticate are served by the same certification authority, then the certification path becomes trivial, and the users unwrap each other's certificates directly;

b) if the CAs of the users are arranged in a hierarchy, a user could store the public keys, certificates and reverse certificates of all certification authorities between the user and the root of the DIT. Typically, this would involve the user in knowing the public keys and certificates of only three or four certification authorities. The user would then only require to obtain the certification paths from the common point of trust;

c) if a user frequently communicates with users certified by a particular other CA, that user could learn the certification path to that CA and the return certification path from that CA, making it necessary only to obtain the certificate of the other user itself from the Directory;

d) certification authorities can cross-certify one another by bilateral agreement. The result is to shorten the certification path;

e) if two users have communicated before and have learned one another's certificates, they are able to authenticate without any recourse to the Directory.

In any case, having learned each other's certificates from the certification path, the users shall check the validity of the received certificates.

18.2.1.1 Example

Figure 8 illustrates a hypothetical example of a DIT fragment, where the CAs form a hierarchy. Besides the information shown at the CAs, we assume that each user knows the public key of its certification authority, and its own public and private keys.

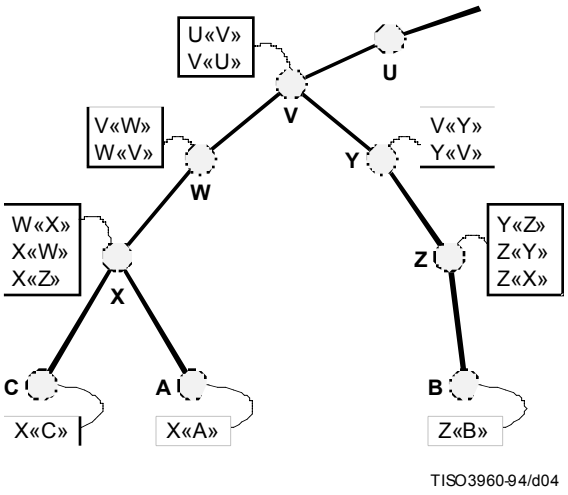


Figure 8 – CA hierarchy – A hypothetical example

If the CAs of the users are arranged in a hierarchy, A can acquire the following certificates from the Directory to establish a certification path to B:

X«W», W«V», V«Y», Y«Z», Z«B»

When A has obtained these certificates, it can unwrap the certification path in sequence to yield the contents of the certificate of B, including Bp:

$$Bp = Xp \bullet X«W» W«V» V«Y» Y«Z» Z«B»$$

In general, A also has to acquire the following certificates from the Directory to establish the return certification path from B to A:

$$Z«Y», Y«V», V«W», W«X», X«A».$$

When B receives these certificates from A, it can unwrap the return certification path in sequence to yield the contents of the certificate of A, including Ap:

$$Ap = Zp \bullet Z«Y» Y«V» V«W» W«X» X«A»$$

Applying the optimizations of 18.2.1:

a) taking A and C, for example: both know Xp, so that A simply has to directly acquire the certificate of C. Unwrapping the certification path reduces to:

$$Cp = Xp \bullet X«C»$$

and unwrapping the return certification Path reduces to:

$$Ap = Xp \bullet X«A»$$

b) assuming that A would thus know W«X», Wp, V«W», Vp, U«V», Up, etc. reduces the information which A has to obtain from the Directory to form the certification path to:

$$V«Y», Y«Z», Z«B»$$

and the information which A has to obtain from the Directory to form the return certification path to:

$$Z«Y», Y«V».$$

c) assuming that A frequently communicates with users certified by Z, it can learn (in addition to the public keys learned in b) above) V«Y», Y«V», Y«Z», and Z«Y». To communicate with B, it need therefore only obtain Z«B» from the Directory.

d) assuming that users certified by X and Z frequently communicate, then X«Z» would be held in the directory entry for X, and vice versa (this is shown in Figure 8). If A wants to authenticate to B, A need only obtain:

$$X«Z», Z«B»$$

to form the certification path, and:

$$Z«X»$$

to form the return certification path.

e) assuming users A and C have communicated before and have learned one another's certificates, they may use each other's public key directly, i.e.

$$Cp = Xp \bullet X«C»$$

and

$$Ap = Xp \bullet X«A»$$

In the more general case the Certification Authorities do not relate in a hierarchical manner. Referring to the hypothetical example in Figure 9, suppose a user D, certified by U, wishes to authenticate to user E, certified by W. The Directory entry of user D shall hold the certificate U«D» and the entry of user E shall hold the certificate W«E».

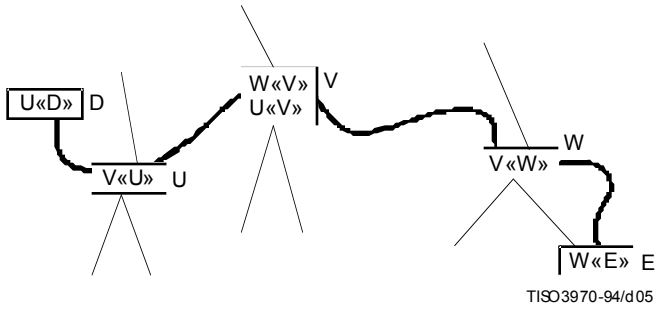


Figure 9 – Non-hierarchical certification path – An example

Let V be a CA with whom CAs U and W have at some previous time exchanged public keys in a trusted way. As a result, certificates U«V», V«U», W«V» and V«W» have been generated and stored in the Directory. Assume U«V» and W«V» are stored in the entry of V, V«U» is stored in U's entry, and V«W» is stored in W's entry.

User D must find a certification path to E. Various strategies could be used. One such strategy would be to regard the users and CAs as nodes, and the certificates as arcs in a directed graph. in these terms, D has to perform a search in the graph to find a path from U to E, one such being U«V», V«W», W«E». When this path has been discovered, the reverse path W«V», V«U», U«D» can also be constructed.

18.2.2 Strong authentication procedures

The basic approach to authentication has been outlined above, namely the corroboration of identity by demonstrating possession of a private key. 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 18.2.2.1, 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 18.2.2.2, 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 18.2.2.3, 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 time stamp checking.

In each case where Strong Authentication is to take place, A must 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 subclause 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.

18.2.2.1 One-way authentication

The following steps are involved, as depicted in Figure 10:

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:

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

   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:

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

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

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

   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.

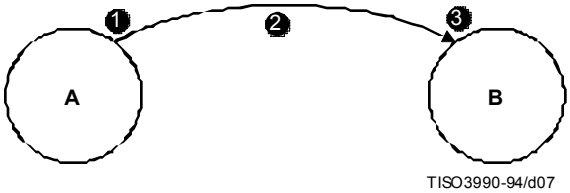


Figure 10 – One-way authentication

18.2.2.2 Two-way authentication

The following steps are involved, as depicted in Figure 11:

1) as for 18.2.2.1;
2) as for 18.2.2.1;
3) as for 18.2.2.1;
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 18.2.2.1, step 3), d).

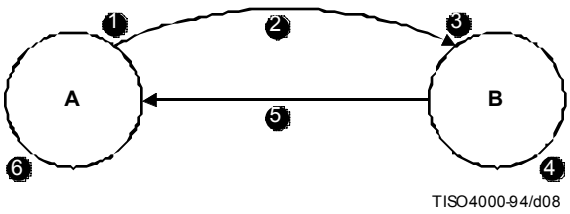


Figure 11– Two-way authentication

18.2.2.3 Three-way authentication

The following steps are involved, as depicted in Figure 12:

1) As for 18.2.2.2.
2) As for 18.2.2.2. Timestamp t^A may be zero.
3) As for 18.2.2.2, except that the timestamp need not be checked.
4) As for 18.2.2.2.
5) As for 18.2.2.2. Timestamp t^B may be zero.
6) As for 18.2.2.2, 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.

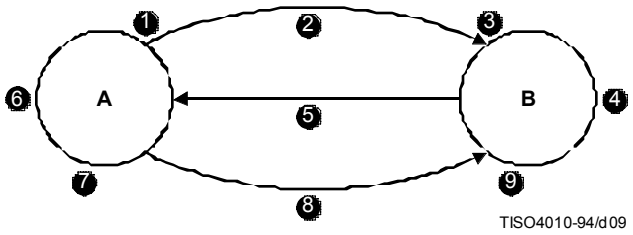


Figure 12– Three-way authentication

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 ITU-T Rec. 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 Specification is used to provide the strong authentication scheme used for this binding.

The Rules Based Access Control (RBAC) scheme defined in ITU-T Rec. X.501 | ISO/IEC 9594-2 makes use of the attribute certificate framework defined in this 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 Specification is used in all Directory protocols defined in this series of Recommendations 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. Privacy protection is 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 their corresponding private key.

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

Annex A

Public-key and Attribute Certificate Frameworks in ASN.1

(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 three ASN.1 modules, **AuthenticationFramework**, **CertificateExtensions,** and **AttributeCertificateDefinitions**.

```
AuthenticationFramework {joint-iso-itu-t ds(5) module(1) authenticationFramework(7) 4}
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
    id-at, id-nf, id-oc, informationFramework, upperBounds, selectedAttributeTypes, basicAccessControl,
    certificateExtensions
        FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1) usefulDefinitions(0) 4}

    Name, ATTRIBUTE, OBJECT-CLASS, NAME-FORM, top
        FROM InformationFramework informationFramework

    ub-user-password, ub-content
        FROM UpperBounds upperBounds

    UniqueIdentifier, octetStringMatch, DirectoryString, commonName
        FROM SelectedAttributeTypes selectedAttributeTypes

    certificateExactMatch, certificatePairExactMatch, certificateListExactMatch, KeyUsage, GeneralNames,

    CertificatePoliciesSyntax, algorithmIdentifierMatch, CertPolicyId
        FROM CertificateExtensions certificateExtensions ;

--public-key certificate definition--

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

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

CertificateSerialNumber ::=     INTEGER

AlgorithmIdentifier             ::=     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   ::=         { ... }

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

SubjectPublicKeyInfo    ::=         SEQUENCE {
        algorithm           AlgorithmIdentifier,
        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
                        -- contains a DER encoding of a value of type &ExtnType
                        -- for the extension object identified by extnId -- }

ExtensionSet EXTENSION          ::=     {  ... }

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

-- other PKI certifiate constructs

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

ForwardCertificationPath        ::=     SEQUENCE OF CrossCertificates

CrossCertificates           ::=         SET OF Certificate

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

CertificatePair     ::=     SEQUENCE {
        issuedToThisCA  [0]     Certificate OPTIONAL,
        issuedByThisCA  [1]     Certificate OPTIONAL
                                -- at least one of the pair shall be present -- }
        (WITH COMPONENTS {..., issuedToThisCA  PRESENT} |
        WITH COMPONENTS {..., issuedByThisCA   PRESENT})


-- certificate revocation list (CRL)
```

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

-- information object classes --

ALGORITHM  ::=  TYPE-IDENTIFIER

-- parameterized types --
HASH {ToBeHashed}       ::=     SEQUENCE {
        algorithmIdentifier         AlgorithmIdentifier,
        hashValue                   BIT STRING ( CONSTRAINED BY {
                -- must 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 {
        -- must be the result of applying a hashing procedure to the DER-encoded (see 6.1) octets --
        -- of a value of -- ToBeSigned -- and then applying an encipherment procedure to those octets -- })

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


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


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


-- PKI object classes --

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


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

```
        ID                      id-oc-pkiCA }


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


cpCps       OBJECT-CLASS        ::= {
        SUBCLASS OF     {top}
        KIND            auxiliary
        MAY CONTAIN     {certificatePolicy |
                        certificationPracticeStmt}
        ID              id-oc-cpCps }

pkiCertPath   OBJECT-CLASS      ::= {
        SUBCLASS OF     {top}
        KIND            auxiliary
        MAY CONTAIN     { pkiPath }
        ID              id-oc-pkiCertPath }


--PKI directory attributes –

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


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

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


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


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


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


supportedAlgorithms  ATTRIBUTE ::= {
        WITH SYNTAX                    SupportedAlgorithm
        EQUALITY MATCHING RULE         algorithmIdentifierMatch
        ID                             id-at-supportedAlgorithms }

SupportedAlgorithm ::= SEQUENCE {
        algorithmIdentifier                    AlgorithmIdentifier,
        intendedUsage                  [0]     KeyUsage OPTIONAL,
        intendedCertificatePolicies[1] CertificatePoliciesSyntax OPTIONAL }


certificationPracticeStmt ATTRIBUTE   ::=       {
        WITH SYNTAX       InfoSyntax
        ID                id-at-certificationPracticeStmt }

InfoSyntax  ::=     CHOICE {
        content     DirectoryString {ub-content},
        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 }

PkiPath    ::=    SEQUENCE OF CrossCertificates


userPassword    ATTRIBUTE    ::=    {
        WITH SYNTAX                     OCTET STRING (SIZE (0..ub-user-password))
        EQUALITY MATCHING RULE          octetStringMatch
        ID                              id-at-userPassword }


-- 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}


END


-- Certificate Extensions module

CertificateExtensions {joint-iso-itu-t ds(5) module(1) certificateExtensions(26) 4}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS ALL --

IMPORTS
        id-at, id-ce, id-mr, informationFramework, authenticationFramework,
                selectedAttributeTypes, upperBounds
                FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1)
                usefulDefinitions(0) 4}

        Name, RelativeDistinguishedName, ATTRIBUTE, Attribute, MATCHING-RULE
                FROM InformationFramework informationFramework

       CertificateSerialNumber, CertificateList, AlgorithmIdentifier,
                EXTENSION, Time, PolicyID
                FROM AuthenticationFramework authenticationFramework

       DirectoryString
                FROM SelectedAttributeTypes selectedAttributeTypes

        ub-name
                FROM UpperBounds upperBounds
```

```
        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),
        nonRepudiation              (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 ::=      { 2 5 29 32 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]     DirectoryString {ub-name} OPTIONAL,
        partyName       [1]     DirectoryString {ub-name} }


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


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 }

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 }

SkipCerts ::= INTEGER (0..MAX)


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

CRLNumber ::= INTEGER (0..MAX)


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 }


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 }


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 {

        distributionPoint               [0] DistributionPointName OPTIONAL,
        onlyContainsUserCerts           [1] BOOLEAN DEFAULT FALSE,
        onlyContainsAuthorityCerts      [2] BOOLEAN DEFAULT FALSE,
        onlySomeReasons                 [3] ReasonFlags OPTIONAL,
        indirectCRL                     [4] BOOLEAN DEFAULT FALSE,
        onlyContainsAttributeCerts      [5] 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 }


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


-- PKI matching rules --
```

```
certificateExactMatch MATCHING-RULE ::= {
        SYNTAX      CertificateExactAssertion
        ID          id-mr-certificateExactMatch }

CertificateExactAssertion ::= SEQUENCE {
        serialNumber        CertificateSerialNumber,
        issuer              Name }


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


    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} )


    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} )


certificateListExactMatch MATCHING-RULE      ::=            {
        SYNTAX      CertificateListExactAssertion
        ID          id-mr-certificateListExactMatch }

CertificateListExactAssertion ::= SEQUENCE {
        issuer              Name,
        thisUpdate          Time,
        distributionPoint   DistributionPointName OPTIONAL }


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 }


algorithmIdentifierMatch MATCHING-RULE      ::=   {
        SYNTAX      AlgorithmIdentifier
        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 }
```

-- 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}


--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}


-- 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

-- Attribute Certificate Framework module
AttributeCertificateDefinitions {joint-iso-itu-t ds(5) module(1) attributeCertificateDefinitions(32) 4}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS ALL --

IMPORTS
        id-at, id-ce, id-mr, informationFramework, authenticationFramework,
                selectedAttributeTypes, upperBounds, id-oc, certificateExtensions
```

```
        FROM UsefulDefinitions {joint-iso-itu-t ds(5) module(1)
        usefulDefinitions(0) 4}

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

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

    DirectoryString, TimeSpecification, UniqueIdentifier
        FROM SelectedAttributeTypes selectedAttributeTypes

    GeneralName, GeneralNames, NameConstraintsSyntax, certificateListExactMatch
        FROM CertificateExtensions certificateExtensions

    ub-name
        FROM UpperBounds upperBounds

    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)}

    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.


-- attribute certificate constructs --

AttributeCertificate ::= SIGNED {AttributeCertificateInfo}

AttributeCertificateInfo ::= SEQUENCE
    {
    version                 AttCertVersion, --version is v2
    holder                  Holder,
    issuer                  AttCertIssuer,
    signature               AlgorithmIdentifier,
    serialNumber            CertificateSerialNumber,
    attrCertValidityPeriod  AttCertValidityPeriod,
    attributes              SEQUENCE OF Attribute,
    issuerUniqueID          UniqueIdentifier OPTIONAL,
    extensions              Extensions      OPTIONAL
    }

AttCertVersion      ::= INTEGER { v2(1) }

Holder      ::=     SEQUENCE
            {
            baseCertificateID   [0] IssuerSerial~~l~~              OPTIONAL,
                -- the issuer and serial number of  the holder's Public Key Certificate
            entityName          [1] GeneralNames            OPTIONAL,
                -- the name of the entity or role
```

```
            objectDigestInfo      [2] ObjectDigestInfo          OPTIONAL
                  --used to directly authenticate the holder, e.g. an executable
--at least one of baseCertificateID, entityName or objectDigestInfo must be present--}

ObjectDigestInfo   ::= SEQUENCE {
      digestedObjectType  ENUMERATED {
            publicKey            (0),
            publicKeyCert        (1),
            otherObjectTypes     (2) },
      otherObjectTypeID          OBJECT IDENTIFIER  OPTIONAL,
      digestAlgorithm            AlgorithmIdentifier,
      objectDigest               BIT STRING }


AttCertIssuer ::=   [0]     SEQUENCE {
    issuerName                   GeneralNames  OPTIONAL,
    baseCertificateID     [0]    IssuerSerial  OPTIONAL,
    objectDigestInfo      [1]    ObjectDigestInfo  OPTIONAL }
-- At least one component must be present
  ( WITH COMPONENTS { ..., issuerName  PRESENT } |
    WITH COMPONENTS { ..., baseCertificateID  PRESENT } |
    WITH COMPONENTS { ..., objectDigestInfo PRESENT } )


IssuerSerial  ::=  SEQUENCE {
      issuer        GeneralNames,
      serial        CertificateSerialNumber,
      issuerUID     UniqueIdentifier OPTIONAL }

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 }


-- 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 }


-- 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 }



--Attribute certificate extensions and matching rules --


attributeCertificateExactMatch MATCHING-RULE  ::= {
        SYNTAX      AttributeCertificateExactAssertion
        ID          id-mr-attributeCertificateExactMatch }

AttributeCertificateExactAssertion       ::=       SEQUENCE {
        serialNumber      CertificateSerialNumber  OPTIONAL,
        issuer            IssuerSerial
        }


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 must 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 }


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


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 }


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}


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


timeSpecificationMatch MATCHING-RULE  ::= {
        SYNTAX      TimeSpecification
        ID          id-mr-timeSpecMatch }


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 }
```

```
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 }


userNotice  EXTENSION  ::= {
        SYNTAX              SEQUENCE SIZE (1..MAX) OF UserNotice
        IDENTIFIED BY       id-ce-userNotice }

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 }

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


acceptablePrivilegePolicies EXTENSION ::= {
        SYNTAX              AcceptablePrivilegePoliciesSyntax
        IDENTIFIED BY       id-ce-acceptablePrivilegePolicies }

AcceptablePrivilegePoliciesSyntax       ::=     SEQUENCE SIZE (1..MAX) OF PrivilegePolicy
```

```
--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}


--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}


--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}


--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}


END
```

Annex B

CRL Generation and Processing Rules

(This annex forms an integral part of this Recommendation |International Standard)

B.1 Introduction

A relying party (certificate user), needs the ability to check the status of a certificate in order to help the decision making as to whether or not to trust that certificate. Certificate Revocation Lists (CRL) are one mechanism for relying parties to use to obtain this information. Other mechanisms may also be used, but are outside the scope of this 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 perform revocation of certificates it issues. Some authorities may combine 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. The **crlScope** extension can be used as one mechanism to determine the scope. This annex provides a mechanism that can be used in the absence of the **crlScope** extension from CRLs.

This annex is written for revocation status checking of public-key certificates using CRLs, EPRLs and 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 purposes of this annex, ACRL can be considered in place of CRL and AARL in place of CARL.

B.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 Certification Authority 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 must 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.

An indirect delta-CRL, EPRL or CARL is a CRL which only contains changes to a set of one or more CRLs, that are complete for their given scopes and in which some or all of those certificates may not have been issued by the authority signing and issuing this CRL.

Within this annex, "Scope of a CRL" is defined by two independent dimensions. One dimension is the set of certificates covered by the CRL. Another dimension is the set of reason codes covered by the CRL. The scope of a CRL can be determined in one or more of the following ways:

- Issuing Distribution Point (IDP) extension in the CRL;
- CRL Scope extension in the CRL;
- Other means, outside the scope of this specification.

B.1.2 CRL processing

If a relying party is using CRLs as the mechanism to determine if a certificate is revoked, they must be sure to use the appropriate CRL(s) for that certificate. This annex describes a procedure for obtaining and processing appropriate CRLs by walking through a number of specific steps. 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 (i.e. revocation status for a certificate) from the given inputs (the certificate itself and input from local policy) is not standardized. For example, although this procedure is described as a sequence of steps to be processed in order, an implementation may use CRLs which are in its local cache rather than retrieving CRLs each time it processes a certificate, providing those CRLs are complete for the scope of the certificate and do not violate any of the parameters of the certificate or policy.

This annex does not include procedures for following pointers in a CRL construct containing the **statusReferrals** extension. 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.

The following general steps are described in B.2 through B.5 below:

1) Determine Parameters for CRLs;
2) Determine CRLs Required;
3) Obtain the CRLs;
4) Process the CRLs.

Step 1 identifies the parameters from the certificate and elsewhere that will be used to determine which types of CRLs are required.

Step 2 applies the values of the parameters to make the determination.

Step 3 identifies the directory attributes from which the CRL types can be retrieved.

Step 4 describes the processing of appropriate CRLs.

B.2 Determine parameters for CRLs

Information located in the certificate itself, as well information from the policy under which the relying party is operating, provide the parameters for determining the appropriateness of candidate CRLs. The following information is required to determine which CRL types are appropriate:

- Certificate type (i.e., end-entity or CA);
- Critical CRL Distribution Point;

- Critical Freshest CRL;
- Reason codes of interest.

The certificate type can be determined from the basic constraints extension in the certificate. If the extension is present, it indicates whether the certificate is a CA-certificate or an end-entity certificate. If the extension is absent, the certificate type is considered to be end-entity. This information is required to determine if a CRL, EPRL or CARL can be used to check the certificate for revocation.

If the certificate contains a critical CRL Distribution Point extension, the relying party certificate processing system must understand this extension in order to be able to trust the certificate. Reliance on a full CRL, for instance, would not be sufficient.

If the certificate contains a critical Freshest CRL extension the relying party cannot use the certificate without first retrieving and checking the freshest CRL.

The reason codes of interest are determined by policy and are generally supplied by the application. It is recommended that these should include all reason codes. This information is required to determine which CRLs are sufficient in terms of reason codes.

Note that policy may also dictate whether or not a relying party is expected to check dCRLs for revocation status, even when the **freshestCRL** extension is flagged non-critical or is absent from the certificate. Though excluded from this step, the processing of these optional dCRLs is described in step 4.

B.3 Determine CRLs required

The values of the parameters described in B.2 determine the criteria upon which the CRL types required to check revocation status of a given certificate is determined. The determination of CRL types can be done based on the following sets of criteria as described in B.3.1 through B.3.4 below.

- End-entity certificate with critical CRL DP asserted;
- End-entity certificate with no critical CRL DP asserted;
- CA-certificate with critical CRL DP asserted;
- CA-certificate with no critical CRL DP asserted.

Handling of the remaining parameters (critical freshest CRL extension and set of reason codes of interest) is done within each of the sub-sections.

Note that in each case, more than one CRL type can satisfy the requirements. Where there is a choice of CRL types, the relying party may select any of the appropriate types to use.

B.3.1 End-entity with critical CRL DP

If the certificate is an end-entity certificate and **cRLDistributionPoints** extension is present in the certificate and flagged critical, the following CRLs shall be obtained:

- A CRL from one of the nominated distribution Point CRLs that covers one or more of the reason codes of interest;
- If all the reason codes of interest are not covered by that CRL, revocation status for the remaining reason codes may be satisfied by any combination of the following CRLs:
  - Additional distribution point CRLs;
  - Additional complete CRLs;
  - Additional complete EPRLs.

If the freshest CRL extension is also present in the certificate and if flagged critical, one or more CRLs shall also be obtained from one or more of the nominated distribution points in that extension, ensuring that freshest revocation information for all reason codes of interest is checked.

B.3.2 End-entity with no critical CRL DP

If the certificate is an end-entity certificate and the **cRLDistributionPoints** extension is absent from the certificate or present and not flagged critical, revocation status for the reason codes of interest may be satisfied by any combination of the following CRLs:

- Distribution point CRLs (if present);
- Complete CRLs;
- Complete EPRLs.

If the freshest CRL extension is also present in the certificate and if flagged critical, one or more CRLs shall also be obtained from one or more of the nominated distribution points in that extension, ensuring that freshest revocation information for all reason codes of interest is checked.

B.3.3 CA with critical CRL DP

If the certificate is a CA and the **cRLDistributionPoints** extension is present in the certificate and flagged critical, the following CRLs/CARLs shall be obtained:

- A CRL or CARL from one of the nominated distribution Points that covers one or more of the reason codes of interest;
- If all the reason codes of interest are not covered by that CRL/CARL, revocation status for the remaining reason codes may be satisfied by any combination of the following CRLs/CARLs:
  - Additional distribution point CRLs/CARLs;
  - Additional complete CRLs;
  - Additional complete CARLs.

If the freshest CRL extension is also present in the certificate and if flagged critical, one or more CRLs/CARLs shall also be obtained from one or more of the nominated distribution points in that extension, ensuring that freshest revocation information for all reason codes of interest is checked.

B.3.4 CA with no critical CRL DP

If the certificate is a CA certificate and the **cRLDistributionPoints** extension is absent from the certificate or present and not flagged critical, revocation status for the reason codes of interest may be satisfied by any combination of the following CRLs:

- Distribution point CRLs/CARLs (if present);
- Complete CRLs;
- Complete CARLs.

If the freshest CRL extension is also present in the certificate and if flagged critical, one or more CRLs/CARLs shall also be obtained from one or more of the nominated distribution points in that extension, ensuring that freshest revocation information for all reason codes of interest is checked.

B.4 Obtain CRLs

If the relying party is retrieving appropriate CRLs from the Directory, these CRLs are obtained from the CRL DP or certificate issuer directory entry by retrieving the appropriate attributes, i.e. one or more of the following attributes:

- Certificate Revocation List;
- Authority Revocation List;
- Delta Revocation List.

B.5 Process CRLs

After considering the parameters discussed in B.2, identifying appropriate CRL types as described in B.3 and retrieving an appropriate set of CRLs as described in B.4, a relying party is ready to process the CRLs. The set of CRLs will contain at least one base CRL and may also contain one or more dCRLs. For each CRL being processed, the relying party must ensure that the CRL is accurate with respect to its scope. The relying party has already determined that the CRL is appropriate for the scope of the certificate of interest, through the process of B.2 and B.3 above. In addition, validity checks must be conducted on the CRLs and they must be checked to determine whether or not the certificate has been revoked. These checks are described in B.5.1 through B.5.4 below.

B.5.1 Validate base CRL scope

As described in B.3, there can be more than one type of CRL that can be used as the base CRL for checking revocation status of a certificate. Depending on the policy of issuing authority with respect to CRL issuance, the relying party may have one or more of the following base CRL types available to them:

- Complete CRL for all entities;
- Complete EPRL;
- Complete CARL;
- Distribution Point Based CRL/EPRL/CARL.

B.5.1.1 – B.5.1.4 provide the set of conditions which must be true in order for a relying party to use a CRL of each type as the base CRL for certificate revocation status checking for reason codes of interest.

Indirect base CRLs are addressed within each of the subclauses.

B.5.1.1 Complete CRL

In order to determine that a CRL is a complete CRL for end-entity and CA-certificates, for all reason codes of interest, the following must be true:

- Delta CRL indicator extension must be absent; and
- Issuing distribution point extension may be present; and
- Issuing distribution point extension must not contain distribution point field; and
- Issuing distribution point extension must not contain **onlyContainsUserCerts** field set to **TRUE**; and
- Issuing distribution point extension must not contain **onlyContainsAuthorityCerts** field set to **TRUE**; and
- Issuing distribution point extension must not contain **onlyContainsAttributeCerts** field set to **TRUE**; and
- If the **reasonCodes** field is present in the issuing distribution point extension, the reasons code field must include all the reasons of interest to the application.

Issuing distribution point extension may or may not contain **indirectCRL** field. Hence, this field need not be checked.

B.5.1.2 Complete EPRL

In order to determine that a CRL is a complete EPRL for reason codes of interest, all of the following must be true:

- Delta CRL indicator extension must be absent; and
- Issuing distribution extension must be present; and
- Issuing distribution point extension must not contain distribution point field; and
- Issuing distribution point extension must contain **onlyContainsUserCerts** field. This field must be set to **TRUE**; and
- Issuing distribution point extension must not contain **onlyContainsAuthorityCerts** field set to **TRUE**; and
- Issuing distribution point extension must not contain **onlyContainsAttributeCerts** field set to **TRUE**; and

- If the **reasonCodes** field is present in the issuing distribution point extension, the reasons code field must include all the reasons of interest to the application; and
- Issuing distribution point extension may or may not contain **indirectCRL** field (hence, this field need not be checked).

This CRL may be only used if the relying party has determined the subject certificate to be an end entity certificate. Thus, for version 3 certificates, if the subject certificate contains the **basicConstraints** extension, its value must be **cA=FALSE**.

B.5.1.3 Complete CARL

In order to determine that a CRL is a complete CARL for reason codes of interest, all of the following conditions must be true:

- Delta CRL indicator extensions must be absent; and
- Issuing distribution must be present; and
- Issuing distribution point must not contain distribution point field; and
- Issuing distribution point must not contain **onlyContainsUserCerts** field set to **TRUE**; and
- Issuing distribution point extension must not contain **onlyContainsAttributeCerts** field set to **TRUE**; and
- Issuing distribution point must contain **onlyContainsAuthorityCerts** field. This field must be set to **TRUE**; and
- If the **reasonCodes** field is present in the issuing distribution point extension, the reasons code field must include all the reasons of interest to the application; and
- Issuing distribution point may or may not contain **indirectCRL** field (hence, this field need not be checked).

This CARL may be only used if the subject certificate is a CA-certificate. Thus, for version 3 certificates, the subject certificate must contain the **basicConstraints** extension with the value **cA=TRUE**.

B.5.1.4 Distribution point based CRL/EPRL/CARL

In order to determine that a CRL is one of the CRLs indicated by a CRL distribution point extension in the certificate, all of the following conditions must be true:

- Either the distribution point field in the CRL's issuing distribution point extension must be absent (only when not looking for a critical CRL DP), or one of the names in the distribution point field in the CRL distribution point extension of the certificate must match one of the names in the distribution point field in the issuing distribution point extension of the CRL. Alternatively, one of the names in the **crlIssuer** field of the certificate's CRL DP can match one of the names in DP of the IDP; and
- If the certificate is an end entity certificate, the CRL must not contain **onlyContainsAuthorityCerts** field set to **TRUE** in the issuing distribution point extension of the CRL; and
- If **onlyContainsAuthorityCerts** is set to **TRUE** in the issuing distribution point extension of the CRL, then the certificate being checked must contain **basicConstraints** extension with **cA** component set to **TRUE**; and
- If the reasons code field is present in the CRL distribution point extension of the certificate, this field must be either absent from the issuing distribution point extension of the CRL or contain at least one of the reason codes asserted in the CRL distribution point extension of the certificate; and
- If the **cRLIssuer** field is absent from the CRL distribution point extension of the certificate, the CRL must be signed by the same CA that signed the certificate; and
- If the **cRLIssuer** field is present in the CRL distribution point extension of the certificate, the CRL must be signed by the CRL Issuer identified in the CRL distribution point extension of the certificate and the CRL must contain the **indirectCRL** field in the issuing distribution point extension.

B.5.2 Validate delta CRL scope

The relying party may also be checking dCRLs, either because required to through a critical **freshestCRL** extension in the certificate or because the policy under which the relying party is operating supports dCRL checking.

A relying party can always be sure that it has the appropriate CRL information for a certificate if all of the following conditions are met:

- The base CRL the relying party is using is appropriate for the certificate (in terms of the scope); and
- The delta CRL the relying party is using is appropriate for the certificate (in terms of the scope); and
- The base CRL was issued at the time or later than the base CRL referenced by the dCRL.

In order to determine that the dCRL is appropriate for the certificate, all of the following conditions must be true:

- Delta CRL indicator extension must be present; and
- The dCRL must be issued after the base CRL. (One way to ensure this is to check that the CRL number in the **crlNumber** extension of the dCRL is greater than the CRL number in the **crlNumber** extension of the base CRL the relying party is using and the **cRLStreamIdentifier** fields in the base and the dCRL match. This approach may require additional logic to account for number wrapping. Another way is to compare the **thisUpdate** fields in the base and dCRLs the relying party has; and
- The base CRL the relying party is using must be the one the dCRL is issued for or a later one. (One way to ensure this is to check that the CRL number in the **deltaCRLIndicator** extension of the dCRL is less than or equal to the CRL number in the **crlNumber** extension of the base CRL the relying party is using and the **cRLStreamIdentifier** fields in the base and the dCRL match. This approach may require additional logic to account for number wrapping. Another way is to compare the **thisUpdate** fields of the base CRL the relying party has and the base CRL pointed to by the dCRL). Yet another way is to compare the **thisUpdate** field in the base CRL the relying party has and the **baseUpdateTime** extension in the dCRL the relying party has; and

  NOTE — a relying party can always construct a base CRL by applying a dCRL to a base CRL as long as the above two rules are satisfied using the **crlNumber** and **cRLStreamIdentifier** checks. In that case, the new base CRL's **crlNumber** extension and **thisUpdate** field are those of the dCRL. The relying party does not know the **nextUpdate** field of the new base CRL and does not need to know for the purpose of associating it with another dCRL.
- If the dCRL contains an Issuing Distribution Point extension, than the scope of the issuing distribution point shall be consistent with the certificate as described in B.5.1.4 above; and
- If the dCRL contains CRL Scope extension, than the certificate shall be within the scope of the CRL; and
- If the dCRL does not contain any of the following extensions: **streamIdentifier**, **crlScope**, and **issuingDistributionPoint**, it shall be used only in conjunction with a full and complete base CRL.

B.5.3 Validity and currency checks on the base CRL

In order to verify that a base CRL is accurate and has not been modified since its issuance, all of the following conditions must be satisfied:

- The relying party must be able to obtain the public key of the issuer identified in the CRL using authenticated means; and
- The signature on the base CRL must be verified using this authenticated public key; and
- If the **nextUpdate** field is present, the current time should be prior to the **nextUpdate** field; and
- The issuer name in the CRL must match the issuer name in the certificate that is being checked for revocation, unless the CRL is retrieved from the CRL DP in the certificate and the CRL DP extension contains the CRL issuer component. In that case, one of the names in CRL issuer component in the CRL DP extension must match issuer name in the CRL.

B.5.4 Validity and checks on the delta CRL

In order to verify that a dCRL is accurate and has not been modified since its issuance, all of the following conditions must be satisfied:

- The relying party must be able to obtain the public key of the issuer identified in the CRL using authenticated means; and
- The signature on the dCRL must be verified using this authenticated public key; and
- If the **nextUpdate** field is present, the current time should be prior to the **nextUpdate** field; and
- The issuer name in the dCRL must match the issuer name in the certificate which is being checked for revocation, unless one of the following conditions is true:
  - Delta CRL is retrieved from the CRL DP in the certificate and the CRL DP extension contains the CRL issuer component. In that case, one of the names in CRL issuer component in the CRL DP extension must match issuer name in the CRL; or
  - Delta CRL is retrieved from the **freshestCRL** in the certificate and the CRL Scope extension in the dCRL contains a Per Authority Scope component with an **authorityName** that matches the issuer name of the certificate.

Annex C
Examples of Delta CRL Issuance

(This annex does not form an integral part of this Recommendation | International Standard)

C.1 Introduction

There are two models for issuing CRLs involving the use of dCRLs for a given set of certificates.

In the first model, each dCRL references the most recent CRL that is complete for the given scope. Several dCRLs may be issued for that same scope before a new CRL that is complete for that scope is issued. The new CRL that is complete for that scope is used as the foundation for the next sequence of dCRLs and is the CRL that is referenced in the relevant extension in the dCRL. When the new CRL that is complete for the scope is issued, a final dCRL for the previous CRL that is complete for the scope is also issued.

The second model, while very similar, differs in that the CRL referenced by a dCRL is not necessarily one that is complete for given scope (i.e. the referenced CRL may only have been issued as a dCRL). If the referenced CRL is one that is complete for the given scope, it may not necessarily be the most recent one that is complete for that scope.

A certificate using system that is processing a dCRL must also have a CRL that is complete for the given scope and that is at least as current as the CRL referenced in the dCRL. This CRL that is complete for the given scope may be one that was issued as such by the responsible authority or it may be one constructed locally by the certificate using system. Note that in some situations there may be duplicate information in the dCRL and CRL that is complete for the given scope if, for example, the certificate using system has a CRL that was issued after the one referenced in the dCRL.

The following table illustrates three examples of the use of dCRLs. Example 1 is the traditional scheme described as the first model above. Examples 2 and 3 are variants of the second model described above.

In example 2, the authority issues CRLs, that are complete for the given scope, every second day and the dCRLs reference the second to last complete for scope CRL. This scheme may be useful in environments where there is a need to reduce the number of users accessing a repository at the same time to retrieve a CRL that is complete for a given scope. In example 2, users that have the most recent CRL that is complete for the scope as well as users who have the second most recent complete for scope CRL can use the same dCRL. Both sets of users have complete revocation information about the certificates for that given scope at the time of issuance of the dCRL being used.

In example 3, CRLs that are complete for the given scope are issued once a week as in example 1, but every dCRL references a base of revocation information 7 days prior to that dCRL.

An example of the use of indirect CRLs is not provided here, but is a superset of those examples.

These are examples only and other variants are also possible, depending on local policy. Some factors which might be considered when establishing that policy include: number of users and frequency accessing CRLs, replication of CRLs, load balancing for directory systems holding CRLs, performance, latency requirements, etc.

Day	Example 1 – Delta references most recent CRL that is complete for given scope		Example 2 – Delta references second most recent CRL that is complete for given scope		Example 3 – Delta references 7 day old revocation information	
	CRL complete for given scope	**Delta-CRL**	**CRL complete for given scope**	**Delta-CRL**	**CRL complete for given scope**	**Delta-CRL**
8	thisUpdate=day 8 nextUpdate=day 15 crlNumber=8	thisUpdate=day 8 nextUpdate=day 9 crlNumber=8 BaseCRLNumber=1	thisUpdate=day 8 nextUpdate=day 10 crlNumber=8	thisUpdate=day 8 nextUpdate=day 9 crlNumber=8 BaseCRLNumber=6	thisUpdate=day 8 nextUpdate=day 15 cRLNumber=8	thisUpdate=day 8 nextUpdate=day 9 cRLNumber=8 BaseCRLNumber= 1
9	not issued	thisUpdate=day 9 nextUpdate=day 10 crlNumber=9 BaseCRLNumber=8	not issued	thisUpdate=day 9 nextUpdate=day 10 crlNumber=9 BaseCRLNumber=6	not issued	thisUpdate=day 9 nextUpdate=day 10 cRLNumber=9 BaseCRLNumber= 2
10	not issued	thisUpdate=day 10 nextUpdate=day 11 crlNumber=10 BaseCRLNumber=8	thisUpdate=day 10 nextUpdate=day 12 crlNumber=10	thisUpdate=day 10 nextUpdate=day 11 crlNumber=10 BaseCRLNumber=8	not issued	thisUpdate=day 10 nextUpdate=day 11 cRLNumber=10 BaseCRLNumber= 3
11-14	Patterns continue as for previous days					
15	thisUpdate=day 15 nextUpdate=day 22 crlNumber=15	thisUpdate=day 15 nextUpdate=day 16 crlNumber=15 BaseCRLNumber=8	not issued	thisUpdate=day 15 nextUpdate=day 16 crlNumber=15 BaseCRLNumber=12	thisUpdate=day 15 nextUpdate=day 22 cRLNumber=15	thisUpdate=day 15 nextUpdate=day 16 cRLNumber=15 BaseCRLNumber= 8
16	Not issued	thisUpdate=day 16 nextUpdate=day 17 crlNumber=16 BaseCRLNumber=15	thisUpdate=day 16 nextUpdate=day 18 crlNumber=16	thisUpdate=day 16 nextUpdate=day 17 crlNumber=16 BaseCRLNumber=14	not issued	thisUpdate=day 16 nextUpdate=day 17 cRLNumber=16 BaseCRLNumber= 9

Annex D
Privilege Policy and Privilege Attribute Definition Examples

(This annex does not form an integral part of this Recommendation | International Standard)

D.1 Introduction

Privilege policy defines, for privilege management, 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. Formal specification of privilege policy can aid a privilege verifier with automated assessment of a privilege asserter's privileges against the sensitivity of the requested resource, since it includes the rules for determining the pass/fail of a privilege asserter's request, given their privilege and the sensitivity of the resource.

Because there are requirements to ensure the integrity of the privilege policy being used in those determinations, an identifier of the privilege policy, in the form of an object identifier, and a HASH of the entire privilege policy can be carried in signed objects, stored in directory entries etc. However, no specific syntax used to define an instance of privilege policy is standardized in this Specification.

D.2 Sample syntaxes

Privilege policy may be defined using any syntax, including plain text. In order to assist those defining privilege policies in understanding various options for the definitions, two sample syntaxes, which could be used for this purpose, are provided in this annex. It must be stressed that these are examples only and implementation of privilege management through the use of attribute certificates or the **subjectDirectoryAttributes** extension of public-key certificates are NOT required to support these or any other specific syntaxes.

D.2.1 First example

The following ASN.1 syntax is one example of a comprehensive and flexible tool for the definition of privilege policy.

```
PrivilegePolicySyntax        ::= SEQUENCE {
   version              Version,
   ppe                  PrivPolicyExpression }

PrivPolicyExpression     ::= CHOICE {
   ppPredicate       [0] PrivPolicyPredicate,
   and               [1] SET SIZE (2..MAX) OF PrivPolicyExpression,
   or                [2] SET SIZE (2..MAX) OF PrivPolicyExpression,
   not               [3] PrivPolicyExpression,
   orderedPPE        [4] SEQUENCE OF PrivPolicyExpression }
   -- note:  "sequence" defines the temporal order in which the
   -- privilege must be examined

PrivPolicyPredicate      ::= CHOICE {
   present                  [0] PrivilegeIdentifier,
   equality                 [1] PrivilegeComparison, -- single/set-valued priv.
   greaterOrEqual           [2] PrivilegeComparison, -- single-valued priv.
   lessOrEqual              [3] PrivilegeComparison, -- single-valued priv.
   subordinate              [4] PrivilegeComparison, -- single-valued priv.
   substrings               [5] SEQUENCE {      -- single-valued priv.
            type                        PrivilegeType,
            initial             [0]     PrivilegeValue OPTIONAL,
            any                 [1]     SEQUENCE OF PrivilegeValue,
            final               [2]     PrivilegeValue OPTIONAL },
   subsetOf                 [6]  PrivilegeComparison, -- set-valued priv.
   supersetOf               [7]  PrivilegeComparison, -- set-valued priv.
   nonNullSetInter          [8] PrivilegeComparison, -- set-valued priv.
   approxMatch              [9] PrivilegeComparison,
```

```
   -- single/set-valued priv. (approximation defined by application)
   extensibleMatch          [10] SEQUENCE {
            matchingRule                OBJECT IDENTIFIER,
            inputs                      PrivilegeComparison } }

PrivilegeComparison     ::= CHOICE {
   explicit             [0]     Privilege,
   -- the value(s) of an external privilege identified by
   -- Privilege.privilegeId is(are) compared with the value(s)
   -- explicitly provided in Privilege.privilegeValueSet
   byReference      [1]     PrivilegeIdPair }
   -- the value(s) of an external privilege identified by
   -- PrivilegeIdPair.firstPrivilege is(are) compared with
   -- the value(s) of a second external privilege identified by
   -- PrivilegeIdPair.secondPrivilege

Privilege           ::= SEQUENCE {
   type                 PRIVILEGE.&id ({SupportedPrivileges}),
   values               SET SIZE (0..MAX) OF
                        PRIVILEGE.&Type ({SupportedPrivileges} {@type})}

SupportedPrivileges  PRIVILEGE ::=      { ... }

PRIVILEGE  ::=      ATTRIBUTE
--Privilege is analogous to Attribute

PrivilegeIdPair         ::= SEQUENCE {
   firstPrivilege           PrivilegeIdentifier,
   secondPrivilege          PrivilegeIdentifier }

PrivilegeIdentifier     ::= CHOICE {
   privilegeType            [0] PRIVILEGE.&id ({SupportedPrivileges}),
   xmlTag                   [1] OCTET STRING,
   edifactField             [2] OCTET STRING }
--PrivilegeIdentifer extends the concept of AttributeType to other
--(e.g., tagged) environments, such as XML and EDIFACT

Version             ::= INTEGER { v1(0) }
```

A concrete example may help to clarify the creation and use of the above **PrivilegePolicy** construct.

Consider the privilege to approve a salary increase. For simplicity, assume that the policy to be enforced states that only Senior Managers and above can approve increases, and that an approval can only be given for a position lower than your own (e.g., a Director can approve an increase for a Senior Manager, but not for a Vice President). For this example, assume that there are six possible ranks ("TechnicalStaff" = 0, "Manager" = 1, "Senior Manager" = 2, "Director" =3, "Vice President" = 4, "President" = 5).

Assume, furthermore, that the Attribute Type (the "privilege") identifying rank in an Attribute Certificate is OBJECT ID *OID-C* and that the Attribute Type (the "sensitivity") identifying rank in the database record whose salary field is to be modified is OBJECT ID *OID-D* (these would of course be replaced by real object identifiers in an actual implementation). The following Boolean expression denotes the desired "salary approval" policy (codifying this in a **PrivilegePolicy** expression is a relatively straightforward task):

```
AND ( NOT ( lessOrEqual ( value corresponding to OID-C, value corresponding to OID-D ) )
      subsetOf ( value corresponding to OID-C, { 2, 3, 4, 5 } ) )
```

This policy encoding says that the rank of the approver must be greater than (expressed as "NOT less-than-or-equal-to") the rank of the approvee AND that the rank of the approver must be one of {Senior Manager, ..., President} in order for this Boolean expression to evaluate to TRUE. The first privilege comparison is "by reference", comparing the values corresponding to the Attribute Type "rank" for both entities involved. The second privilege comparison is "explicit"; here the value corresponding to the privilege "rank" for the approver is compared with an explicitly-included list of values. The privilege verifier in this situation, therefore, needs a construct codifying this policy along with two Attributes, one associated with the approver and one associated with the approvee. The approver's Attribute (which would be contained in an Attribute Certificate) may have the value {*OID-C* 3} and the approvee's Attribute (which may be contained in a database record) may have the value {*OID-D* 3}. Comparing the AttributeValue corresponding to the approver's Attribute Type (in this example, 3) with the Attribute Value corresponding to the approvee's Attribute Type (in this example, also 3) results in a FALSE for the "NOT lessOrEqual" expression, and so the first Director is denied the ability to approve a salary increase for the second Director. On the other hand, if the approvee's Attribute was {*OID-D* 1}, the Director would be granted the ability to approve the Manager's increase.

It is not difficult to conceive of useful additions to the above expression. For example, a third component of the **'and'** could be added saying that the environment variable "currentTime", read from the local clock and then encoded as an attribute of type OBJECT ID *OID-E*, must be within a particular span specified explicitly in the expression as an attribute of type OBJECT ID *OID-F*. Thus, for example, salary updates may be permitted only if the above conditions are satisfied and the request takes place during business hours.

D.2.2 Second example

A security policy in its simplest form is a set of criteria for the provision of security services. With regard to access control, security policy is a subset of a higher system-level security policy that defines the means for enforcing access control policies between initiators and targets. The access control mechanisms must: permit communication where a specific policy permits; deny communication where a specific policy does not explicitly permit.

A security policy is the basis for the decisions made by the access control mechanisms. Domain-specific security policy information is conveyed via the Security Policy Information File (SPIF).

The SPIF is a signed object to protect it from unauthorized changes. The SPIF contains information used to interpret the access control parameters contained in the security label and the clearance attribute. The security policy identifier that appears in the clearance attribute must be associated with a specific implementation syntax and semantics as defined by the security policy. This implementation syntax associated with a specific security policy is maintained in a SPIF.

The SPIF conveys equivalencies between authorizations and sensitivities across security policy domains as determined by security policies; provides a printable representation of security labels; and maps displayable strings to security levels and categories for presentation to end users when selecting a data object's security attributes. Equivalency mappings are expressed such that a label generated under one security policy domain may be correctly interpreted by an application operating in another security policy domain. The SPIF also maps the clearance attribute to the message security label fields and the presentation labels that should be displayed to the user. This mapping, if successful, verifies that the intended recipient has the proper authorizations to accept the data object.

A SPIF contains a sequence of the following:

- **versionInformation** – indicates the version of the ASN.1 syntax.
- **updateInformation** – indicates the version of the syntax and semantics of the SPIF specification.
- **securityPolicyIdData** - identifies the security policy to which the SPIF applies.
- **privilegeId** – indicates the OID that identifies the syntax that is included in the clearance attribute security category.
- **rbacId** - object identifier which identifies the syntax of the security category that is used in conjunction with the SPIF.
- **securityClassifications** - maps the classification of the security label to a classification in the clearance attribute, and also provides equivalency mappings.

- **securityCategoryTagSets** - maps the security categories of the security label to the security categories in the clearance attribute, and also provides equivalency mappings.
- **equivalentPolicies** – consolidates all equivalent policies in the SPIF.
- **defaultSecurityPolicyIdData**- identifies the security policy which will apply if data is received without a security label.
- **extensions** - provides a mechanism to include additional capabilities as future requirements are identified.

The Security Policy Information File is defined in the following syntax:

```
SecurityPolicyInformationFile ::= SIGNED {SPIF}

SPIF ::=  SEQUENCE    {
        versionInformation                      VersionInformationData DEFAULT v1,
        updateInformation                       UpdateInformationData,
        securityPolicyIdData                    ObjectIdData,
        privilegeId                             OBJECT IDENTIFIER,
        rbacId                                  OBJECT IDENTIFIER,
securityClassifications         [0] SEQUENCE OF SecurityClassification  OPTIONAL,
        securityCategories              [1]     SEQUENCE OF SecurityCategory  OPTIONAL,
        equivalentPolicies              [2]     SEQUENCE OF EquivalentPolicy  OPTIONAL,
        defaultSecurityPolicyIdData     [3]     ObjectIdData OPTIONAL,
        extensions                      [4]     Extensions OPTIONAL }

VersionInformationData ::= INTEGER { v1(0) }

UpdateInformationData  ::=  SEQUENCE    {
        sPIFVersionNumber               INTEGER,
        creationDate                    GeneralizedTime,
        originatorDistinguishedName     Name,
        keyIdentifier                   OCTET STRING OPTIONAL }

ObjectIdData        ::=     SEQUENCE  {
        objectId                OBJECT IDENTIFIER,
        objectIdName            DirectoryString {ubObjectIdNameLength} }

SecurityClassification  ::=  SEQUENCE  {
        labelAndCertValue               INTEGER,
        classificationName              DirectoryString {ubClassificationNameLength},
        equivalentClassifications [0]   SEQUENCE OF EquivalentClassification  OPTIONAL,
        hierarchyValue                  INTEGER,
        markingData               [1]   SEQUENCE OF MarkingData OPTIONAL,
        requiredCategory          [2]   SEQUENCE OF OptionalCategoryGroup  OPTIONAL,
        obsolete                        BOOLEAN DEFAULT FALSE }

EquivalentClassification ::= SEQUENCE {
        securityPolicyId        OBJECT IDENTIFIER,
        labelAndCertValue       INTEGER,
        applied                 INTEGER {
                                        encrypt     (0),
                                        decrypt     (1),
                                        both        (2) } }

MarkingData ::= SEQUENCE {
        markingPhrase   DirectoryString {ubMarkingPhraseLength} OPTIONAL,
        markingCodes    SEQUENCE OF MarkingCode OPTIONAL }

MarkingCode ::= INTEGER {
        pageTop         (1),
        pageBottom      (2),
```

```
        pageTopBottom           (3),
        documentEnd             (4),
        noNameDisplay           (5),
        noMarkingDisplay        (6),
        unused                  (7),
        documentStart           (8),
        suppressClassName (9)  }

OptionalCategoryGroup ::= SEQUENCE {
        operation               INTEGER {
                                        onlyOne         (1),
                                        oneOrMore       (2),
                                        all             (3)},
        categoryGroup           SEQUENCE OF OptionalCategoryData }

OptionalCategoryData ::= SEQUENCE {
        optCatDataId            OC-DATA.&id({CatData}),
        categorydata            OC-DATA.&Type({CatData}{@optCatDataId }) }

OC-DATA ::= TYPE-IDENTIFIER

CatData OC-DATA ::= { ... }

EquivalentPolicy    ::= SEQUENCE {
        securityPolicyId        OBJECT IDENTIFIER,
        securityPolicyName      DirectoryString {ubObjectIDNameLength} OPTIONAL}

Extensions ::= SEQUENCE OF Extension

Extension ::= SEQUENCE {
        extensionId             EXTENSION.&objId ({ExtensionSet}),
        critical                BOOLEAN DEFAULT FALSE,
        extensionValue          OCTET STRING  }
```

Note that the SPIF example is an evolving syntax and full definition and description of each element can be found in SC27 N2315 CD 15816 Security Information Objects.

D.3 Privilege attribute example

The following example of an attribute to convey a particular privilege is provided purely as an illustration only. The actual specification of this syntax and associated attribute is contained in ITU-T Rec X.501 | ISO/IEC 9594-2 clause 17.5. This particular attribute conveys clearance that can be associated with a named entity, including a DUA for purposes of communication with a DSA.

A clearance attribute associates a clearance with a named entity including DUAs.

```
clearance  ATTRIBUTE ::=  {
        WITH SYNTAX         Clearance
        ID                  id-at-clearance }

Clearance  ::=  SEQUENCE {
        policyId            OBJECT IDENTIFIER,
        classList           ClassList DEFAULT {unclassified},
        securityCategories  SET SIZE (1MAX) OF SecurityCategory  OPTIONAL}

ClassList  ::=  BIT STRING {
        unmarked    (0),
        unclassified (1),
        restricted  (2),
        confidential (3),
        secret      (4),
        topSecret   (5) }
```

The individual components are described with the actual specification of this privilege in the referenced document.

Annex E[1)]

An introduction to public key cryptography[2)]

(This annex does not form an integral part of this Recommendation | International Standard)

In conventional cryptographic systems, the key used to encipher information by the originator of a secret message is the same as that used to decipher the message by the legitimate recipient.

In public key cryptosystems (PKCS), however, keys come in pairs, one key of which is used for enciphering and the other for deciphering. Each key pair is associated with a particular user X. One of the keys, known as the public key (Xp) is publicly known, and can be used by any user to encipher data. Only X, who possesses the complementary private key (Xs) may decipher the data. (This is represented notationally by D = Xs[Xp[D]]). It is computationally infeasible to derive the private key from knowledge of the public key. Any user can thus communicate a piece of information which only X can find out, by enciphering it under Xp. By extension, two users can communicate in secret, by using each other's public key to encipher the data, as shown in Figure E.1.

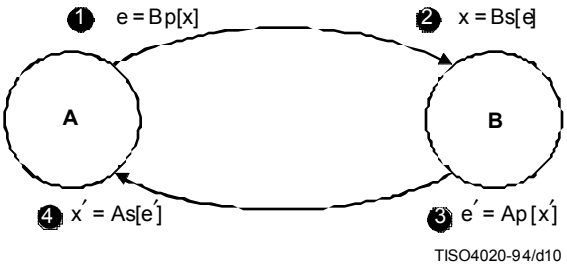


Figure E.1 – Use of a PKCS to exchange secret information

User A has public key Ap and private key As, and user B has another set of keys, Bp and Bs. A and B both know the public keys of each other, but are unaware of the private key of the other party. A and B may therefore exchange secret information with one another using the following steps (illustrated in Figure E.1).

1) A wishes to send some secret information x to B. A therefore enciphers x under B's enciphering key and sends the enciphered information e to B. This is represented by:

$$e = Bp[x]$$

2) B may now decipher this encipherment e to obtain the information x by using the secret decipherment key Bs. Note that B is the only possessor of Bs, and because this key may never be disclosed or sent, it is impossible for any other party to obtain the information x. The possession of Bs determines the identity of B. The decipherment operation is represented by:

$$x = Bs[e], \text{ or } x = Bs[Bp[x]]$$

3) B may now similarly send some secret information, x', to A, under A's enciphering key, Ap:

$$e' = Ap[x']$$

4) A obtains x' by deciphering e':

$$x' = As[e'], \text{ or } x' = As[Ap[x']]$$

---

2) For further information, see:

DIFFIE (W.) and HELLMAN (M.E.): New Directions in Cryptography, *IEEE Transactions on Information Theory*, IT-22, No. 6, (November 1976).

By this means, A and B have exchanged secret information x and x'. This information may not be obtained by anyone other than A and B, providing that their private keys are not revealed.

Such an exchange can, as well as transferring secret information between the parties, serve to verify their identities. Specifically, A and B are identified by their possession of the secret deciphering keys, As and Bs respectively. A may determine if B is in possession of the secret deciphering key, Bs, by having returned part of his information x in B's message x'. This indicates to A that communication is taking place with the possessor of Bs. B may similarly test the identity of A.

It is a property of some PKCS that the steps of decipherment and encipherment can be reversed, as in D=Xp[Xs[D]]. This allows a piece of information which could only have been originated by X, to be readable by any user (who has possession of Xp). This can, therefore, be used in the certifying of the source of information, and is the basis for digital signatures. Only PKCS which have this (permutability) property are suitable for use in this authentication framework. One such algorithm is described in Annex D.

Annex F

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) 4}
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) 4}
        ALGORITHM
                FROM AuthenticationFramework authenticationFramework      ;

-- categories of object identifier --

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

-- object identifier assignments --

id-ea-rsa       OBJECT IDENTIFIER       ::=     {id-ea 1}

-- the following object identifier assignments reserve values assigned to deprecated functions

id-ha-sqMod-n           OBJECT IDENTIFIER  ::=          {id-ha 1}
id-sa-sqMod-nWithRSA    OBJECT IDENTIFIER  ::=          {id-sa 1}

END
```

Annex G

Examples of use of certification path constraints

(This annex does not form an integral part of this Recommendation | International Standard)

G.1 Example 1: Use of basic constraints

Suppose the Widget Corporation wants to cross-certify the central CA of the Acme Corporate Group, but only wants the Widget community to use end-entity certificates issued by that CA, not certificates issued by other CAs certified by that CA.

The Widget Corporation could satisfy this requirement by issuing a certificate for Acme's central CA, including the following extension field value:

Value of Basic Constraints Field:

{ cA TRUE, pathLenConstraint 0 }

G.2 Example 2: Use of name constraints

Suppose the Widget Corporation wants to cross-certify the central CA of the Acme Corporate Group, but only wants the Widget community to use Acme certificates for subjects that meet the following criteria:

- in Acme, Inc. in the U.S., all subjects are acceptable except for subjects in purchasing;
- in EuroAcme in France, only those subjects that are immediately subordinate to the EuroAcme headquarters are acceptable (this includes individuals reporting directly to headquarters but excludes those reporting to subordinate organizations); and
- in Acme Ltd. in the U.K., all subjects are acceptable except those reporting to organizations that are subordinate to the R&D organizational unit (this includes individuals reporting directly to R&D but excludes those reporting to subordinate units of R&D).

The Widget Corporation could satisfy these requirements by issuing a certificate for Acme's central CA, including the following extension field values:

Value of Basic Constraints Field:

{ cA TRUE }

Value of Name Constraints Field:

{ permittedSubtrees {{base *--Country=US, Org=Acme Inc--***},**

{base *--Country=France, Org=EuroAcme--***, maximum 1},**

{base *--Country=UK, Org=Acme Ltd--***}},**

excludedSubtrees {{base *--Country=US, Org=Acme Inc, Org. Unit=Purchasing-***},**

{base *--Country=UK Org=Acme Ltd., Org. Unit=R&D--***, minimum 2}}}**

G.3 Example 3: Use of policy mapping and policy constraints

Suppose the following cross-certification scenario is required between the Canadian and U.S. governments:

a) a Canadian government CA wishes to certify use of U.S. government signatures with respect to a Canadian policy called *Can/US-Trade*;

b) the U.S. government has a policy called *US/Can-Trade*, which the Canadian government is prepared to consider equivalent to its *Can/US-Trade* policy;

c) the Canadian government wants to apply safeguards which require all U.S. certificates to explicitly state support for the policy and which inhibit mapping to other policies within the U.S. domain.

A Canadian government CA could issue a certificate for a U.S. government CA with the following extension field values:

Value of Certificate Policies Field:

{{ policyIdentifier -- *object identifier for Can/US-Trade* -- **}}**

Value of Policy Mappings Field:

{{ issuerDomainPolicy -- *object identifier for Can/US-Trade* -- **,**

subjectDomainPolicy -- *object identifier for US/Can-Trade* -- **}}**

Value of PolicyConstraints Field:

{{ policySet { -- *object identifier for Can/US-Trade* -- **}, requireExplicitPolicy (0),**

inhibitPolicyMapping (0)}}

Annex H

Alphabetical list of information item definitions

(This annex does not form an integral part of this Recommendation | International Standard)

This annex provides an alphabetical index to the definitions of certificate and CRL formats, certificate extensions, object classes, name forms, attribute types and matching rules defined in this Directory Specification.

ITEM	CLAUSE
Certificate and CRL formats	
Attribute certificate format	12.1
Certificate revocation list	7.3
Public-key certificate format	7
Certificate, CRL & CRL entry extensions	
Acceptable certificate policies extension	15.5.2.3
Acceptable privilege policies extension	15.1.2.4
Attribute descriptor extension	15.3.2.2
Authority attribute identifier extension	15.5.2.4
Authority key identifier extension	8.2.2.1
Base update extension	8.6.2.5
Basic attribute constraints extension	15.5.2.1
Basic constraints extension	8.4.2.1
Certificate issuer extension	8.6.2.3
Certificate policies extension	8.2.2.6
CRL distribution points extension	8.6.2.1
CRL number extension	8.5.2.1
CRL scope extension	8.5.2.5
CRL stream identifier extension	8.5.2.7
Delegated name constraints extension	15.5.2.2
Delta CRL indicator extension	8.6.2.4
Delta information extension	8.5.2.9
Extended key usage extension	8.2.2.4
Freshest CRL extension	8.6.2.6
Hold instruction code extension	8.5.2.3
Inhibit any policy extension	8.4.2.4

Invalidity date extension	8.5.2.4
Issuer alternative name extension	8.3.2.2
Issuing distribution point extension	8.6.2.2
Key usage extension	8.2.2.3
Name constraints extension	8.4.2.2
No revocation information extension	15.2.2.2
Ordered list extension	8.5.2.8
Policy constraints extension	8.4.2.3
Policy mappings extension	8.2.2.7
Private key usage period extension	8.2.2.5
Reason code extension	8.5.2.2
Role specification certificate identifier extension	15.4.2.1
SOA identifier extension	15.3.2.1
Status referral extension	8.5.2.6
Subject alternative name extension	8.3.2.1
Subject key identifier extension	8.2.2.2
Subject directory attributes extension	8.3.2.3
Targeting information extension	15.1.2.2
Time specification extension	15.1.2.1
User notice extension	15.1.2.3
Object classes and name forms	
Attribute certificate CRL distribution point object class	17.1.4
Certificate policy and CPS object class	11.1.5
CRL distribution points object class and name form	11.1.3
Delta CRL object class	11.1.4
PKI CA object class	11.1.2
PKI certificate path object class	11.1.6
PKI user object class	11.1.1
PMI AA object class	17.1.2
PMI delegation path	17.1.5
PMI SOA object class	17.1.3
PMI user object class	17.1.1

Privilege policy object class	17.1.6
Directory attributes	
AA certificate attribute	17.2.2
AA certificate revocation list attribute	17.2.5
Attribute certificate attribute	17.2.1
Attribute certificate revocation list attribute	17.2.4
Attribute descriptor certificate attribute	17.2.3
Authority revocation list attribute	11.2.5
CA certificate attribute	11.2.2
Certification practice statement attribute	11.2.8
Certificate policy attribute	11.2.9
Certificate revocation list attribute	11.2.4
Cross certificate pair attribute	11.2.3
Delegation path attribute	17.2.6
Delta revocation list attribute	11.2.6
PKI path attribute	11.2.10
Privilege policy attribute	17.2.7
Supported algorithms attribute	11.2.7
User certificate attribute	11.2.1
Matching rules	
AA identifier match	15.5.2.4.1
Acceptable certificate policies match	15.5.2.3.1
Algorithm identifier match	11.3.7
Attribute certificate exact match	17.3.1
Attribute certificate match	17.3.2
Attribute descriptor match	15.3.2.2.1
Basic attribute constraints match	15.5.2.1.1
Certificate exact match	11.3.1
Certificate list exact match	11.3.5
Certificate list match	11.3.6
Certificate match	11.3.2
Certificate pair exact match	11.3.3

Certificate pair match	11.3.4
Delegated name constraints match	15.5.2.2.1
Delegation path match	17.3.4
Holder issuer match	17.3.3
PKI Path match	11.3.9
Policy match	11.3.8
Role specification certificate ID match	15.4.2.1.1
Time specification match	15.1.2.1.1

Annex I

Amendments and corrigenda

(This annex does not form an integral part of this Recommendation | International Standard)

This edition of this Directory Specification includes the following amendments:

– Amendment 1 for Certificate Extensions;

This edition of this Directory Specification includes the following technical corrigenda correcting the defects in the following defect reports (some parts of some of the following Technical Corrigenda may have been subsumed by the amendments that formed this edition of this Directory Specification):

– Technical Corrigendum 1 (covering Defect Reports 183 and 194);

– Technical Corrigendum 3 (covering Defect Reports 200, 201, 212, 213, 218, and 220);

– Technical Corrigendum 4 (covering Defect Report 185);

– Technical Corrigendum 5 (covering Defect Report 204);

– Technical Corrigendum 7 (covering Defect Report 222).