INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.509
(08/97)

SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

Directory

Information technology – Open Systems Interconnection – The Directory: Authentication framework

ITU-T Recommendation X.509

(Previously CCITT Recommendation)

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

PUBLIC DATA NETWORKS	
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEM INTERCONNECTION	
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300–X.349
Satellite data transmission systems	X.350–X.399
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	**X.500–X.599**
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600–X.629
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	
Systems Management framework and architecture	X.700–X.709
Management Communication Service and Protocol	X.710–X.719
Structure of Management Information	X.720–X.729
Management functions and ODMA functions	X.730–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	
Commitment, Concurrency and Recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999

For further details, please refer to ITU-T List of Recommendations.

INTERNATIONAL STANDARD 9594-8

ITU-T RECOMMENDATION X.509

INFORMATION TECHNOLOGY – OPEN SYSTEMS INTERCONNECTION – THE DIRECTORY: AUTHENTICATION FRAMEWORK

Summary

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.

Source

The ITU-T Recommendation X.509 was approved on the 9th of August 1997. The identical text is also published as ISO/IEC International Standard 9594-8.

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. Some commonly known threats, together with the security services and mechanisms that can be used to protect against them, are briefly described in Annex B. 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 the provision of authentication services by the Directory to its users. These users include the Directory itself, as well as other applications and services. The Directory can usefully be involved in meeting their needs for authentication and other security services because it is a natural place from which communicating parties can obtain authentication information of each other – knowledge which is the basis of authentication. The Directory is a natural place because it holds other information which is required for communication and obtained prior to communication taking place. Obtaining the authentication information of a potential communication partner from the Directory is, with this approach, similar to obtaining an address. Owing to the wide reach of the Directory for communications purposes, it is expected that this authentication framework will be widely used by a range of applications.

This third edition technically revises and enhances, but does not replace, the second edition of this Recommendation | International Standard. Implementations may still claim conformance to the second edition. However, at some point, the second edition will not be supported (i.e. reported defects will no longer be resolved). It is recommended that implementations conform to this third edition as soon as possible.

This third edition specifies versions 1 and 2 of the Directory protocols.

The first and second editions also specified version 1. Most of the services and protocols specified in this edition are designed to function under version 1. When version 1 has been negotiated, differences between the services and between the protocols defined in the three editions are accommodated using the rules of extensibility defined in the 1997 edition of ITU-T Rec. X.519 | ISO/IEC 9594-5. However, some enhanced services and protocols, e.g. signed errors, will not function unless all Directory entites involved in the operation have negotiated version 2.

Implementors should note that a defect resolution process exists and that corrections may be applied to this Recommendation | International Standard in the form of technical corrigenda. The identical corrections will be applied to this Recommendation | International Standard in the form of an Implementor's Guide. A list of approved technical corrigenda for this Recommendation | International Standard can be obtained from the Subcommittee secretariat. Published technical corrigenda are available from your national standards organization. The Implementor's Guide may be obtained from the ITU Web site.

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 authentication framework.

Annex B, which is not an integral part of this Recommendation | International Standard, describes security requirements.

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

Annex D, which is not an integral part of this Recommendation | International Standard, describes the RSA public key cryptosystem.

Annex E, which is not an integral part of this Recommendation | International Standard, describes hash functions.

Annex F, which is not an integral part of this Recommendation | International Standard, describes threats protected against by the strong authentication method.

Annex G, which is not an integral part of this Recommendation | International Standard, describes data confidentiality.

Annex H, 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 I, which is not an integral part of this Recommendation | International Standard, contains a bibliography.

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

Annex K, 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: AUTHENTICATION FRAMEWORK

SECTION 1 – GENERAL

1 Scope

This Recommendation | International Standard:

- 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.

The strong authentication method specified in this Recommendation | International Standard is based upon public key cryptosystems. It is a major advantage of such systems that user certificates may be held within the Directory as attributes, and may be freely communicated within the Directory System and obtained by users of the Directory in the same manner as other Directory information. The user certificates are assumed to be formed by ‘off-line’ means, and may subsequently be placed in the Directory. The generation of user certificates is performed by some off-line Certification Authority which is completely separate from the DSAs in the Directory. In particular, no special requirements are placed upon Directory providers to store or communicate user certificates in a secure manner.

A brief introduction to public key cryptography can be found in Annex C.

In general, the authentication framework is not dependent on the use of a particular cryptographic algorithm, provided it has the properties described in clause 7. Potentially a number of different algorithms may be used. 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 will serve to maximize the community of users able to authenticate and communicate securely. One example of a public key cryptographic algorithm can be found in Annex D.

Similarly, two users wishing to authenticate shall support the same hash function (see 3.3.14) (used in forming credentials and authentication tokens). Again, in principle, a number of alternative hash functions could be used, at the cost of narrowing the communities of users able to authenticate. A brief introduction to hash functions can be found in Annex E.

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 (1995) | ISO/IEC 10021-4:1997, *Information technology – Message Handling Systems (MHS): Message transfer system: Abstract service definition and procedures*.
- ITU-T Recommendation X.500 (1997) | ISO/IEC 9594-1:1997, *Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services*.
- ITU-T Recommendation X.501 (1997) | ISO/IEC 9594-2:1997, *Information technology – Open Systems Interconnection – The Directory: Models*.
- ITU-T Recommendation X.511 (1997) | ISO/IEC 9594-3:1997, *Information technology – Open Systems Interconnection – The Directory: Abstract service definition.*
- ITU-T Recommendation X.518 (1997) | ISO/IEC 9594-4:1997, *Information technology – Open Systems Interconnection – The Directory: Procedures for distributed operation.*
- ITU-T Recommendation X.519 (1997) | ISO/IEC 9594-5:1997, *Information technology – Open Systems Interconnection – The Directory: Protocol specifications.*
- ITU-T Recommendation X.520 (1997) | ISO/IEC 9594-6:1997, *Information technology – Open Systems Interconnection – The Directory: Selected attribute types.*
- ITU-T Recommendation X.521 (1997) | ISO/IEC 9594-7:1997, *Information technology – Open Systems Interconnection – The Directory: Selected object classes.*
- ITU-T Recommendation X.525 (1997) | ISO/IEC 9594-9:1997, *Information technology – Open Systems Interconnection – The Directory: Replication*.
- ITU-T Recommendation X.530 (1997) | ISO/IEC 9594-10:1997, *Information technology – Open Systems Interconnection – The Directory: Use of systems managemnt for administration of the Directory*.
- CCITT Recommendation X.660 (1992) | ISO/IEC 9834-1:1993, *Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: General procedures*.
- ITU-T Recommendation X.680 (1994) | ISO/IEC 8824-1:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*
- ITU-T Recommendation X.681 (1994) | ISO/IEC 8824-2:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification.*
- ITU-T Recommendation X.682 (1994) | ISO/IEC 8824-3:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.*
- ITU-T Recommendation X.683 (1994) | ISO/IEC 8824-4:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications.*

– ITU-T Recommendation X.690 (1994) | ISO/IEC 8825-1:1995, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*

– ITU-T Recommendation X.690[1] Cor. 2 | ISO/IEC 8825-1[1] Cor. 2.

– ITU-T Recommendation X.880 (1994) | ISO/IEC 13712-1:1995, *Information technology – Remote Operations: Concepts, model and notation.*

– ITU-T Recommendation X.881 (1994) | ISO/IEC 13712-2:1995, *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*.

2.3 Other references

– ISO/IEC 11770-1:1996, *Information technology – Security techniques – Key management – Part 1: Framework.*

– POSTEL (J.B.), Internet Protocol, RFC 791, *Internet Activities Board*, 1981.

– CROCKER (D.H.), Standard for the Format of ARPA Internet Text Messages, RFC 822, *Internet Activities Board*, 1982.

– MOCKAPETRIS (P.), Domain Names – Implementation and Specification, RFC 1035, *Internet Activities Board*, 1987.

– BERNERS-LEE (T.), Universal Resource Identifiers in WWW, RFC 1630, *Internet Activities Board*, 1994.

3 Definitions

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

3.1 OSI Reference Model security architecture definitions

The following terms are defined in CCITT Rec. X.800 and 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).

[1] Presently at the stage of draft.

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 Authentication framework definitions

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

3.3.1 attribute certificate: A set of attributes of a user together with some other information, rendered unforgeable by the digital signature created using the private key of the certification authority which issued it.

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

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

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

3.3.5 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.

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

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

3.3.8 certification authority: An authority trusted by one or more users to create and assign certificates. Optionally the certification authority may create the users' keys.

3.3.9 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.

3.3.10 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.

3.3.11 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.

3.3.12 delta-CRL: A partial CRL indicating only changes since a prior CRL issue.

3.3.13 end entity: A certificate subject which uses its public key for purposes other than signing certificates.

3.3.14 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.

3.3.15 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).

3.3.16 one-way function: A (mathematical) function f which is easy to compute, but which for a general value y in the range, 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.

3.3.17 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.

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

3.3.19 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.

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

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

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

3.3.23 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 the authentication framework is to describe the relationship between an authenticating entity and a certification authority; an authenticating entity shall be certain that it can trust the certification authority to create only valid and reliable certificates.

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

4 Abbreviations

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

CA	Certification Authority
CRL	Certificate Revocation List
DIB	Directory Information Base
DIT	Directory Information Tree
DSA	Directory System Agent
DUA	Directory User Agent
PKCS	Public Key Cryptosystem

5 Conventions

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

The term "Directory Specification" (as in "this Directory Specification") shall be taken to mean this Recommendation | International Standard. The term "Directory Specifications" shall be taken to mean all of the X.500-Series Recommendations | ISO/IEC 9594.

This Directory Specification uses the term "1988 edition systems" to refer to systems conforming to the first (1988) 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. Systems conforming to this third edition of the Directory Specifications are referred to as "1997 edition systems".

This Directory Specification presents ASN.1 notation in the bold Times Roman, 9 point typeface. When ASN.1 types and values are referenced in normal text, they are differentiated from normal text by presenting them in the bold Times Roman, 9 point typeface. The names of procedures, typically referenced when specifying the semantics of processing, are differentiated from normal text by displaying them in bold Helvetica. Access control permissions are presented in italicized Helvetica.

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 • 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.	

SECTION 2 – SIMPLE AUTHENTICATION

6 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 signalling of procedures for protecting passwords may be a matter for extension to this Recommendation | International Standard.

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 1.

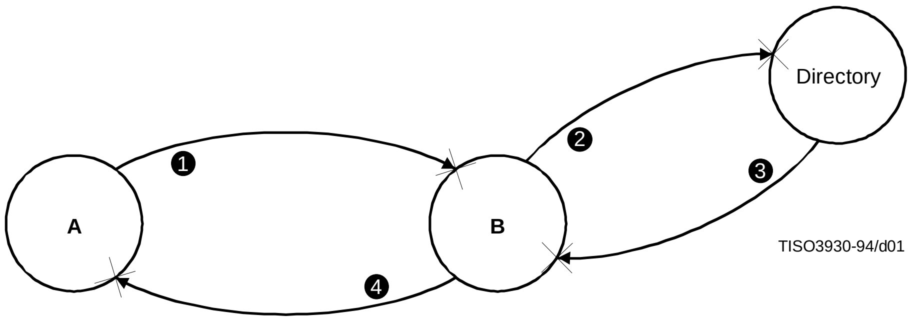


Figure 1 – The unprotected simple authentication procedure

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);

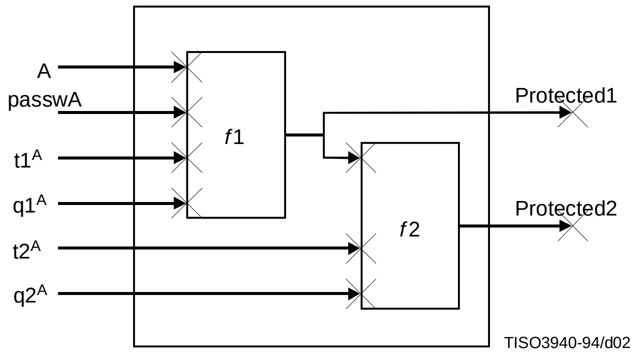
3) the Directory confirms (or denies) to B that the credentials are valid;

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).

6.1 Generation of protected identifying information

Figure 2 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 2 – Protected simple authentication

6.2 Procedure for protected simple authentication

Figure 3 illustrates the procedure for protected simple authentication.

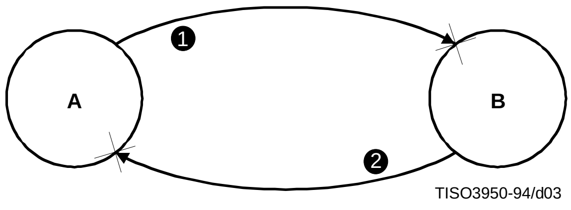


Figure 3 – The protected simple authentication procedure

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 2, where the timestamp and/or random number (when used) is used to minimize replay and to conceal the password.

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

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

   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 2. The further protection is of the form:

   Protected2 = *f*2 ($t2^A$, $q2^A$, 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 [this is similar in principle to step 1) of 6.2].

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

NOTE – The procedures defined in these items 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.

6.3 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 }
```

SECTION 3 – STRONG AUTHENTICATION

7 Basis of strong authentication

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 C 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. One example of a cryptographic algorithm can be found in Annex D.

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 of doing this: clause 8 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.

8 Obtaining a user's public key

In order for a user to trust the authentication procedure, it shall obtain the other user's public key from a source that it trusts. Such a source, called a Certification Authority (CA), uses the public key algorithm to certify the public key, producing a *certificate*. 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 clause 9) 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:

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

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.

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 a canonical 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.

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.

Specific extensions may be defined in Recommendations | International Standards or by any organization which has a need. The object identifier which identifies an extension shall be defined in accordance with CCITT Rec. X.660 | ISO/IEC 9834-1. Standard extensions for certificates are defined in clause 12.

The following object class is used to define specific extensions.

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

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 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 $\rightarrow$ B):

– starts with a certificate produced by CA(A), namely CA(A)«X^1» for some entity X^1;

– continues with further certificates X^i«X^{i+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 $\rightarrow$ B and B $\rightarrow$ 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.

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 as follows:

```
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 }

CertificatePair                     ::=          SEQUENCE {
       forward                 [0]  Certificate OPTIONAL,
       reverse                 [1]  Certificate OPTIONAL
                                           -- at least one of the pair shall be present -- }
```

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                   ForwardCertificationPath 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.

```
ForwardCertificationPath       ::=  SEQUENCE OF CrossCertificates

CrossCertificates              ::=  SET OF Certificate
```

8.1 Optimization of the amount of information obtained from the Directory

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.

8.2 Example

Figure 4 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.

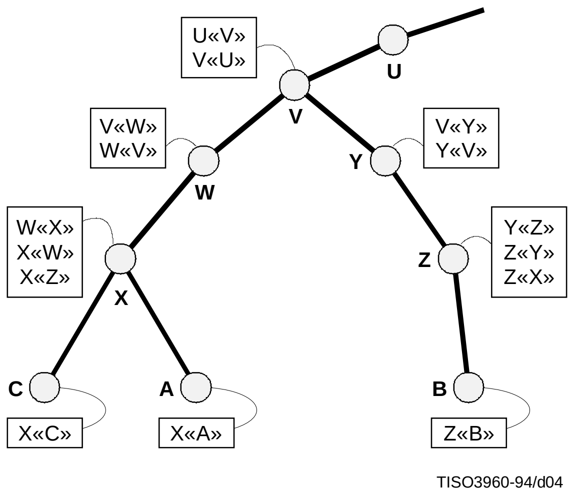


Figure 4 – CA hierarchy – A hypothetical example

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 • Z«Y» Y«V» V«W» W«X» X«A»

Applying the optimizations of 8.1:

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 • X«C»

   and unwrapping the return certification path reduces to:

   Ap = Xp • 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 4). 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 • X«C»

   and

   Ap = Xp • X«A»

In the more general case, the Certification Authorities do not relate in a hierarchical manner. Referring to the hypothetical example in Figure 5, 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».

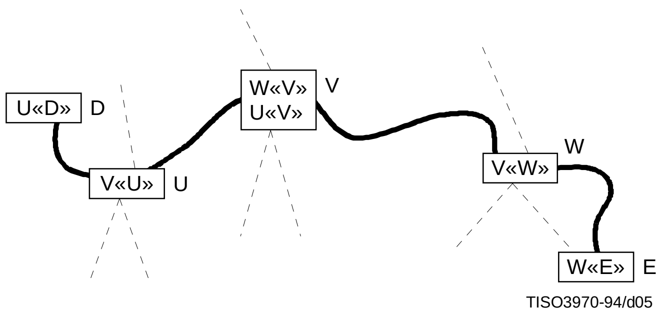


Figure 5 – Non-hierarchical certification path – An example

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.

9 Digital signatures

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 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 6). Thus:

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

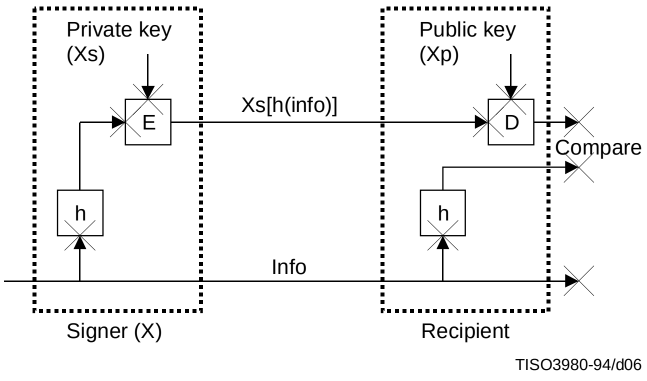


Figure 6 – Digital signatures

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 authentication framework 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 | SO/IEC 8825-1) 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:

```
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 BasicEncoding 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 ISO 8825, 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 bits 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/Cor. 2 | ISO/IEC 8825-1/Cor. 2;

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

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.

10 Strong authentication procedures

10.1 Overview

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 Recommendations | International 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 10.2, 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 10.3, 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 10.4, involves, in addition, a further transfer from A to B. It establishes the same properties as the two-way authentication, but does so without the need for association timestamp checking.

In each case where Strong Authentication is to take place, A 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 clause 7 above. Any such access is not mentioned again in the description of the procedures below.

The checking of timestamps as mentioned in the following subclauses 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.

10.2 One-way authentication

The following steps are involved, as depicted in Figure 7:

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:

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

   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:

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

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

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

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 $B \rightarrow A$, 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.

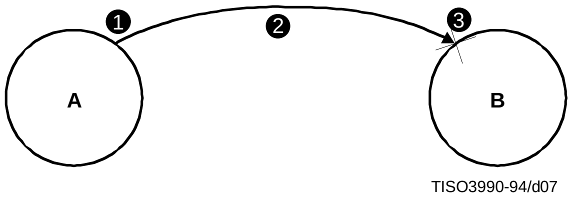


Figure 7 – One-way authentication

10.3 Two-way authentication

The following steps are involved, as depicted in Figure 8:

1) as for 10.2;

2) as for 10.2;

3) as for 10.2;

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:

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

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

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

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 10.2, step 3, d)].

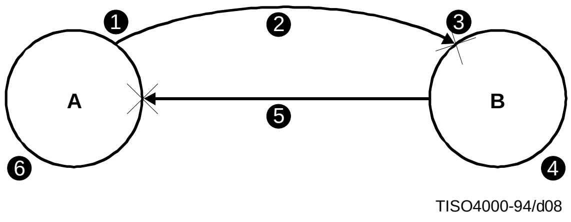


Figure 8 – Two-way authentication

10.4 Three-way authentication

The following steps are involved, as depicted in Figure 9:

1) As for 10.3.
2) As for 10.3. Timestamp t^A may be zero.
3) As for 10.3, except that the timestamp need not be checked.
4) As for 10.3.
5) As for 10.3. Timestamp t^B may be zero.
6) As for 10.3, 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.

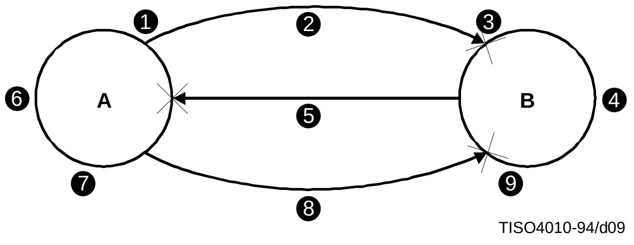


Figure 9 – Three-way authentication

11 Management of keys and certificates

11.1 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 the authentication 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 as described in Annex D.

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.

11.2 Management of certificates

A 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.

The production of a certificate occurs off-line and shall not be performed with an automatic query/response mechanism. The advantage of this certification is that because the private key of the certification authority, CAs, is never known except in the isolated and physically secure CA, the CA private key may then only be learnt by an attack on CA itself, making compromise unlikely.

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 11.1 b) or of 11.1 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 11.1 a) or of 11.1 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 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.

Certificates shall have a lifetime associated with them, at the end of which they expire. In order to provide continuity of service, the CA shall ensure timely availability of replacement certificates to supersede expired/expiring certificates. 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 CA 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 CA 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 CA, or if the CA's certificate is assumed to be compromised. Four related points are:

– The revocation of a user certificate or CA certificate shall be made known by the CA, and a new certificate shall be made available, if appropriate. The CA may then inform the owner of the certificate about its revocation by some off-line procedure.

– The CA shall maintain:

  a) a timestamped list of the certificates it issued which have been revoked;

  b) a timestamped list of revoked certificates of all CAs known to the CA, certified by the CA.

  Both certified lists shall exist, even if empty.

– The maintenance of Directory entries affected by the CA'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.

– The revocation lists ("black-lists") are held within entries as attributes of types "CertificateRevocationList" and "AuthorityRevocationList". These attributes can be operated on using the same operations as other attributes. These attribute types are defined as follows:

```
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 }

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 {
            userCertificate             CertificateSerialNumber,
            revocationDate              Time,
            crlEntryExtensions    Extensions OPTIONAL } OPTIONAL,
      crlExtensions [0]           Extensions OPTIONAL }}
```

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 CA, e.g. in that CA's policy.

NOTE 2 – If a non-repudiation of data service is dependent on keys provided by the CA, the service should ensure that all relevant keys of the CA (revoked or expired) and the 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 shall be absent. This may permit an 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 may 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 performs 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 12.

12 Certificate and CRL extensions

12.1 Introduction

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. 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.

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.

12.2 Key and policy information

12.2.1 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.

12.2.2 Certificate and CRL extension fields

The following extension fields are defined:

a) *Authority key identifier;*

b) *Subject key identifier;*

c) *Key usage;*

d) *Private key usage period;*

e) *Certificate policies;*

f) *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.

In addition, 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.

12.2.2.1 Authority key identifier field

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 12.3.2.1 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.

This extension is always non-critical.

12.2.2.2 Subject key identifier field

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.

12.2.2.3 Key usage field

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 non-repudiation 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 bits **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. It is an advisory field and does not imply that usage of the key is restricted to the purpose indicated. 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.

12.2.2.4 Extended key usage field

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 CCITT 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. It is an advisory field and does not imply that usage of the key is restricted by the certification authority to the purpose 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.

12.2.2.5 Private key usage period field

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.

12.2.2.6 Certificate policies field

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. Typically, different certificate policies will relate to different applications which may use the certified key. 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 12.4.3). 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. 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.

12.2.2.7 Policy mappings field

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.

This extension is always non-critical.

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.

12.2.2.8 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 12.2.2.3 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.

12.3 Certificate subject and certificate issuer attributes

12.3.1 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.

12.3.2 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.

12.3.2.1 Subject alternative name field

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.

12.3.2.2 Issuer alternative name field

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.

12.3.2.3 Subject directory attributes field

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.

12.4 Certification path constraints

12.4.1 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 this Recommendation | International Standard.

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:

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

b) 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.

12.4.2 Certificate extension fields

The following extension fields are defined:

a) *Basic constraints*;

b) *Name constraints*;

c) *Policy constraints*.

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 L.

12.4.2.1 Basic constraints field

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, 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 that system should use other means to determine if the certified public key may 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.

12.4.2.2 Name constraints field

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, then a certificate-using system shall check that the certification path being processed is consistent with the value in this extension.

12.4.2.3 Policy constraints field

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, it indicates that, in all certificates starting from that issued by a nominated CA in the certification path until the end of the certification path, it is necessary for the certificate 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 or the identifier of a policy which has been declared equivalent to it through policy mapping. The nominated CA is either the subject 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.

12.4.3 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.

The inputs to the certification path processing procedure are:

a) a set of certificates comprising a certification path;

b) a trusted public key value or key identifier (if the key is stored internally to the certification path processing module), for use in verifying the first 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; and

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

The values of c), d), and e) will depend upon the policy requirements of the user-application combination that needs to use the certified end-entity public key.

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) If validation was successful, a set of policies constrained by the CAs in the certification path, together with all qualifiers for these policies encountered in the certification path, or the special value *any-policy*. Unless *any-policy* is returned, the certificate user shall only use the certificate in accordance with one of the identified policies and shall process all qualifiers for that policy present in the certification path.

d) If validation was successful and c) returned the value *any-policy*, the set of all policy element qualifiers encountered in the certification path.

e) 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.

The procedure makes use of the following set of state variables:

a) *user-constrained-policy-set*: A set of certificate policy identifiers comprising the policies currently considered acceptable to the certificate user; this state variable can also take the special value *any-policy*.

b) *authority-constrained-policy-set*: A set of certificate policy identifiers comprising the set of policies currently considered acceptable to the CAs in the certification path; this state variable can also take the special value *any-policy*.

c) *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*.

d) *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.

e) *explicit-policy-indicator*: Indicates if an acceptable policy needs to be explicitly identified in every certificate.

f) *policy-mapping-inhibit-indicator*: Indicates if policy mapping is inhibited.

g) *pending-constraints*: Details of explicit-policy and/or inhibit-policy-mapping constraints which have been stipulated but are yet to take effect. There are two one-bit indicators called *explicit-policy-pending* and *policy-mapping-inhibit-pending* together with, for each, an integer called *skip-certificates* which gives the number of certificates yet to skip before the constraint takes effect.

The procedure involves an initialization step, followed by a series of certificate-processing steps. The initialization step comprises:

a) initialize the *user-constrained-policy-set* variable to the value of *initial-policy-set*;

b) initialize the *authority-constrained-policy-set* variable to the value *any-policy*;

c) initialize the *permitted-subtrees* variable to *unbounded*;

d) initialize the *excluded-subtrees* variable to an empty set;

e) initialize the *explicit-policy-indicator* to the *initial-explicit-policy* value;

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

g) initialize the two *pending-constraints* indicators to unset.

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*.

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.

c) If *explicit-policy-indicator* is set and *user-constrained-policy-set* is not set to *any-policy*, check that the certificate policies extension is present and that at least one member of *user-constrained-policy-set* appears in the certificate policies field.

d) If the certificate policies extension is present and is flagged critical, compute the intersection of the policies in that extension and the *authority-constrained-policy-set* and put the result as the new value of *authority-constrained-policy-set.* Check that the intersection of *authority-constrained-policy-set* and *user-constrained-policy-set* is non-empty.

e) 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*.

If any of the above checks fails, the procedure terminates, returning a failure indication and an appropriate reason code. If none of the above checks fails on the end certificate, the procedure terminates, returning a success indication together with the set of policy identifiers from *authority-constrained-policy-set*, the required policy element qualifiers, and details of any policy mapping that may have occurred.

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 *explicit-policy-indicator* is not set:

   – if the *explicit-policy-pending* indicator is set, 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 *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).

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

   – process any policy mapping extension with respect to policies in the *user-constrained-policy-set* and add appropriate policy identifiers to the *user-constrained-policy-set*;

   – process any policy mapping extension with respect to policies in the *authority-constrained-policy-set* and add appropriate policy identifiers to the *authority-constrained-policy-set*;

   – if the *policy-mapping-inhibit-pending* indicator is set, 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).

12.5 Basic CRL extensions

12.5.1 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 12.6) 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 a CA 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 CA posted the revocation. Further information may be known as to when an actual or suspected key compromise occurred, and this information may be valuable to a 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.

12.5.2 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*.

The CRL number field shall be used only as a CRL extension field and the other fields shall be used only as CRL entry extension fields.

12.5.2.1 CRL number field

This CRL extension field conveys a monotonically increasing sequence number for each CRL issued by a given CRL issuer through a given CA 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.

12.5.2.2 Reason code field

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) }
```

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.

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 12.5.2.3). 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 12.6) 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 contains an entry for the same certificate with reason code **certificateHold**.

This extension is always non-critical.

12.5.2.3 Hold instruction code field

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".

12.5.2.4 Invalidity date field

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 CA processed the revocation. This field is defined as follows:

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

This extension is always non-critical.

NOTE 1 –The date in this extension is not, by itself, sufficient for non-repudiation purposes. For example, this date may be a date advised by the private key holder, and it is possible for such a person 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 a CA 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.

12.6 CRL distribution points and delta-CRLs

12.6.1 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 CAs. 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 delta-CRL, 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 delta-CRLs:

a) In order to control CRL sizes, it needs to be possible to assign subsets of the set of all certificates issued by one CA 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 certficiates, e.g. a certification path. This can be achieved by having one CRL issuer sign and issue CRLs containing revocations from multiple CAs.

c) There is a requirement for separate CRLs covering revoked CA certificates and revoked end-entity certificates. This facilitates processing of certification paths as the CRL for revoked CA 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 delta-CRLs) which only contain entries for certificates that have been revoked since the issuance of a base CRL.

f) While requirements a) through e) may lead to optional revocation mechanisms, every CA shall, as a common fall-back approach, issue complete CRLs covering all certificates it issues.

12.6.2 Certificate extension fields

The CRL distribution points extension shall be used only as a certificate extension and may be used in both CA certificates and end-entity 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 can obtain a current complete CRL from the CA 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) }
```

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 CA'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 the CRL distribution point extension is made critical, the CA shall ensure that the distribution points cover reason codes **keyCompromise** and/or **cACompromise** whichever is applicable.

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, at a minimum, the reason code **keyCompromise** (for an end-entity certificate) or **cACompromise** (for a CA certificate).

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 CA (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 – 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 CA is an aspect of CA policy.

12.6.3 CRL and CRL entry extension fields

The following CRL or CRL entry extension fields are defined:

a) *Issuing distribution point*;

b) *Certificate issuer*;

c) *Delta CRL indicator*.

Issuing distribution point and delta CRL indicator shall be used only as CRL extensions. Certificate issuer shall be used only as a CRL entry extension.

12.6.3.1 Issuing distribution point field

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 CA 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. 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,
        onlyContainsCACerts        [2] BOOLEAN DEFAULT FALSE,
        onlySomeReasons            [3] ReasonFlags OPTIONAL,
        indirectCRL                [4] BOOLEAN DEFAULT FALSE }
```

The **distributionPoint** component contains the name of the distribution point in one or more name forms. If this field is absent, the CRL shall contain entries for all revoked unexpired certificates issued by the CRL issuer.

If **onlyContainsUserCerts** is true, the CRL only contains revocations for end-entity certificates. If **onlyContainsCACerts** is true, the CRL only contains revocations for CA 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 CAs other than the issuer of the CRL. The particular CA 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 12.6.3.2. 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**, **onlyContainsCACerts**, and **onlySomeReasons** indicators, from all CAs that identify this CRL issuer in their certificates.

For CRLs distributed via the Directory, the following rules regarding use of attributes apply. A CRL which has **onlyContainsCACerts** 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 CA 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 CA. CRLs not containing critical extensions must contain all current CRL entries for the issuing CA, including entries for all revoked user certificates and CA certificates.

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

NOTE 2 – If a CA issues, from its own directory entry (i.e. not from a separately-named CRL distribution point), a CRL with **onlyContainsUserCerts** or **onlyContainsCACerts** set, then the CA should ensure that all certificates covered by this CRL contain the **basicConstraints** extension.

12.6.3.2 Certificate issuer field

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.

12.6.3.3 Delta CRL indicator field

This CRL extension field identifies a CRL as being a delta-CRL only. 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 starting point in the generation of this delta-CRL, i.e. this delta-CRL contains the changes between the base CRL and the complete CRL issued along with this delta-CRL. It is the decision of a CA as to whether to provide delta-CRLs. However, a delta-CRL shall not be issued without a corresponding complete CRL being issued at the same time. The value of the CRL number, as conveyed in the CRL number extension field (if present), shall be identical for both the delta-CRL and the corresponding complete CRL.

The CRL user constructing a locally held CRL from delta-CRLs shall consider the constructed CRL incomplete and unusable if both the following conditions are true:

– the value of the **CRLNumber** of the received delta-CRL is more than one greater than that of the **CRLNumber** of the delta-CRL last processed; and

– the value of **BaseCRLNumber** of the received delta-crl has changed from the **BaseCRLNumber** of the delta-CRL last processed.

This extension is always critical. A certificate user that does not understand the use of delta-CRLs should not use a CRL containing this extension, as the CRL may not be as complete as the user expects. CRLs not containing critical extensions must contain all current CRL entries for the issuing CA, including entries for all revoked user certificates and CA certificates.

NOTE – It is the decision of the CA as to whether or not to distribute the CRLs issued between two base CRLs. For example, it may be the policy of the CA to distribute base CRLs via CD-ROM and delta-CRLs via an on-line service such as the Directory. Although the CA had issued its CRL with the associated delta-CRL, it may be the CA's policy that the user shall construct the current CRL by applying the delta-CRL to the base CRL held on the CD-ROM.

12.6.4 Attribute type for delta-CRLs

The following attribute type is defined for holding a delta-CRL in a CA directory entry:

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

12.7 Matching rules

This Directory Specification defines matching rules for use with attributes of type **Certificate**, **CertificatePair**, **CertificateList**, and **SupportedAlgorithm**, respectively.

12.7.1 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.

12.7.2 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 }

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 (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:

a) **serialNumber** matches if the value of this component in the attribute value equals that in the presented value;

b) **issuer** matches if the value of this component in the attribute value equals that in the presented value;

c) **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;

d) **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;

e) **certificateValid** matches if the presented value falls within the validity period of the stored attribute value;

f) **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;

g) **subjectPublicKeyAlgID** matches if it is equal to the **algorithm** component of the **algorithmIdentifier** of the **subjectPublicKeyInformation** component of the stored attribute value;

h) **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;

i) **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;

j) **policy** matches if all of the policy elements identified in one of the presented values are contained in the set of **policyElementIds** in any of the **policyInformation** values in the certificate policies extension in the stored attribute value; there is no match if there is no certificate policies extension in the stored attribute value;

k) **pathToName** matches unless the certificate has a name constraints extension which inhibits the construction of a certification path to the presented name value.

12.7.3 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 {
        forwardAssertion  [0] CertificateExactAssertion OPTIONAL,
        reverseAssertion  [1] CertificateExactAssertion OPTIONAL }
        ( WITH COMPONENTS      {..., forwardAssertion PRESENT} |
          WITH COMPONENTS      {..., reverseAssertion PRESENT} )
```

This matching rule returns TRUE if the components that are present in the **forwardAssertion** and **reverseAssertion** components of the presented value match the corresponding components of the **forward** and **reverse** components, respectively, in the stored attribute value.

12.7.4 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 forward or reverse certificate of the pair.

```
certificatePairMatch MATCHING-RULE ::=       {
        SYNTAX              CertificatePairAssertion
        ID          id-mr-certificatePairMatch }

CertificatePairAssertion ::= SEQUENCE {
        forwardAssertion    [0] CertificateAssertion OPTIONAL,
        reverseAssertion    [1] CertificateAssertion OPTIONAL }
        ( WITH COMPONENTS      {..., forwardAssertion PRESENT} |
          WITH COMPONENTS      {..., reverseAssertion PRESENT} )
```

This matching rule returns TRUE if all of the components that are present in the **forwardAssertion** and **reverseAssertion** components of the presented value match the corresponding components of the **forward** and **reverse** components, respectively, in the stored attribute value, using the rules given in 12.7.2.

12.7.5 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.

12.7.6 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 }
```

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:

a) **issuer** matches if the value of this component in the attribute value equals that in the presented value;

b) **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;

c) **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;

d) **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 no match if the stored attribute value contains no issuing distribution point extension;

e) **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;

f) **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.

12.7.7 Algorithm identifier match

The algorithm identifier match rule compares for equality a presented value with an attribute value of type **SupportedAlgorithm**.

```
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.

13 Obtaining certified attributes

Users identities are bound to their public key certificates for authentication and identification purposes. Each public key certificate conveys the information necessary to perform certain cryptographic functions. Certified attributes associated with a subject, such as clearances, may be conveyed in a separate structure, defined as an attribute certificate (**AttributeCertificate**).

13.1 Attribute certificates

An attribute certificate is a separate structure from a subject's X.509 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 Certification Authority create both the public key certificate and attribute certificate(s) for a user; in fact, separation of duties will frequently dictate otherwise. However, the implications of performing multiple certificate path validations should be considered when defining local policy. Exchanges between a subject and the Certification Authority dealing with the generation of attribute certificates are outside the scope of this Recommendation | International Standard.

```
AttributeCertificate ::= SIGNED {AttributeCertificateInfo}

AttributeCertificateInfo ::= SEQUENCE {
        version     Version DEFAULT v1,
        subject     CHOICE {
            baseCertificateID   [0]   IssuerSerial,  -- associated  with a Public Key Certificate
            subjectName         [1]   GeneralNames },  -- associated  with a name
        issuer                  GeneralNames,  -- CA issuing the attribute certificate
        signature               AlgorithmIdentifier,
        serialNumber            CertificateSerialNumber,
        attrCertValidityPeriod  AttCertValidityPeriod,
        attributes              SEQUENCE OF Attribute,
        issuerUniqueID          UniqueIdentifier OPTIONAL,
        extensions              Extensions OPTIONAL }

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

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

The **version** number differentiates between different versions of the attribute certificate. For this Recommendation | International Standard, the version number shall be **v1**.

Within the **subject** field is conveyed the identity of the certificate's subject, referenced either by the **serialNumber** associated with the subject's X.509 public key certificate or by the **subjectName**.

The **issuer** is the name of the Certification Authority who created the attribute certificate.

The **serialNumber** is the serial number that uniquely identifies the attribute certificate.

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

The **attCertValidityPeriod** field conveys the time period during which the attribute certificate is considered valid, expressed in Generalized Time format.

The **attributes** field contains any directory attribute (not restricted to attributes stored within a Directory system, but must be recognized within the domain) concerning the subject.

The **issuerUniqueID** uniquely identifies the issuer of the attribute certificate in those instances where the issuer name is not sufficient.

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

13.2 Attribute certificate attribute

An attribute certificate may be used to certify certain attribute values and bind them to either an X.509 certificate or directly to a subject's name.

```
attributeCertificateAttribute ATTRIBUTE  ::= {
        WITH SYNTAX                     AttributeCertificate
        EQUALITY MATCHING-RULE          certificateExactMatch
        ID                              id-at-attributeCertificate }
```

13.3 Attribute certificate matching rule

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  {
        subject
        [0]   CHOICE {
                        baseCertificateID     [0]  IssuerSerial,
                        subjectName           [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:

a) **baseCertificateID** matches if it is equal to the **IssuerSerial** component of the stored attribute value;

b) **subjectName** matches if the stored attribute value contains the name extension with the same name type as indicated in the presented value;

c) **issuer** matches if the stored attribute value contains the name component of the same name type as indicated in the presented value;

d) **attCertValidity** matches if it falls within the specified validity period of the stored attribute value.

e) for each **attType** in the presented value, there is an attribute of that type present in the **attributes** component of the stored value.

13.4 Attribute certificate paths

A user may obtain one or more attribute certificates from one or more Certificate Authorities. Each attribute certificate bears the name of the Certificate Authority which issued it. 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 }
```

13.5 Attribute certificate revocation list

Attribute certificates may be revoked prior to their expiration time if the attributes are no longer relevant. The attribute certificates may be revoked without impacting the user's public key certificate. The revocation lists are held within entries as an attribute of type **AttributeCertificateRevocationList**. This attribute type is defined as follows:

```
attributeCertificateRevocationList        ATTRIBUTE ::= {
        WITH SYNTAX     CertificateList
        ID              id-at-attributeCertificateRevocationList}
```

NOTE – All of the amendments used for Version 2 Certificate Revocation Lists are applicable to Attribute Certificate Revocation Lists.

Annex A

Authentication framework 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 the two ASN.1 modules **AuthenticationFramework** and **CertificateExtensions**.

```
AuthenticationFramework {joint-iso-ccitt ds(5) module(1) authenticationFramework(7)3}
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, informationFramework, upperBounds, selectedAttributeTypes, basicAccessControl,
        certificateExtensions
            FROM UsefulDefinitions {joint-iso-ccitt ds(5) module(1) usefulDefinitions(0) 3}

    Name, ATTRIBUTE
            FROM InformationFramework informationFramework

    ub-user-password
            FROM UpperBounds upperBounds

    AuthenticationLevel
            FROM BasicAccessControl basicAccessControl

    UniqueIdentifier, octetStringMatch
            FROM SelectedAttributeTypes selectedAttributeTypes

        certificateExactMatch, certificatePairExactMatch, certificateListExactMatch
            FROM CertificateExtensions certificateExtensions ;
-- basic 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 certificate 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 {
            forward         [0]   Certificate OPTIONAL,
            reverse         [1]   Certificate OPTIONAL
                            --  at least one of the pair shall be 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 {
                    userCertificate          CertificateSerialNumber,
                    revocationDate           Time,
                    crlEntryExtensions       Extensions OPTIONAL } OPTIONAL,
            crlExtensions       [0]  Extensions OPTIONAL }}
--  attribute certificate

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

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

attributeCertificate ATTRIBUTE  ::= {
        WITH SYNTAX                          AttributeCertificate
        EQUALITY MATCHING-RULE    attributeCertificateMatch
        ID                                   id-at-attributeCertificate }

AttributeCertificate ::= SIGNED {AttributeCertificateInfo}

AttributeCertificateInfo ::= SEQUENCE {
        version                  Version DEFAULT v1,
        subject  CHOICE {
            baseCertificateID   [0]   IssuerSerial,  --  associated  with a Public Key Certificate
            subjectName          [1]   GeneralNames },  --  associated  with a name
        issuer                       GeneralNames,  --  CA issuing the attribute certificate
        signature                    AlgorithmIdentifier,
        serialNumber                 CertificateSerialNumber,
        attCertValidityPeriod     AttCertValidityPeriod,
        attributes               SEQUENCE OF Attribute,
        issuerUniqueID           UniqueIdentifier OPTIONAL,
        extensions               Extensions OPTIONAL }

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

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

attributeCertificateMatch  MATCHING-RULE  ::=  {
        SYNTAX         AttributeCertificateAssertion
        ID           id-mr-attributeCertificateMatch  }

AttributeCertificateAssertion  ::=  SEQUENCE  {
        subject
        [0]    CHOICE {
                             baseCertificateID     [0]  IssuerSerial,
                             subjectName           [1]  Name} OPTIONAL,
        issuer            [1]    Name OPTIONAL,
        attCertValidity
                          [2]    GeneralizedTime OPTIONAL,
        attType           [3]    SET OF AttributeType OPTIONAL}
--  At least one component of the sequence must be present
--  attribute types --

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

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 }

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

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

attributeCertificateRevocationList        ATTRIBUTE ::= {
            WITH SYNTAX      CertificateList
            ID                  id-at-attributeCertificateRevocationList}
-- information object classes --

ALGORITHM ::=    TYPE-IDENTIFIER
-- parameterized types --

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 -- })

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 }}
-- object identifier assignments --

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-attributeCertificate        OBJECT IDENTIFIER         ::=   {id-at 58}

END

-- Certificate Extensions module

CertificateExtensions {joint-iso-ccitt ds(5) module(1) certificateExtensions(26) 0}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS ALL --

IMPORTS
            id-at, id-ce, id-mr, informationFramework, authenticationFramework,
                  selectedAttributeTypes, upperBounds
                  FROM UsefulDefinitions {joint-iso-ccitt ds(5) module(1)
                  usefulDefinitions(0) 3}
            Name, RelativeDistinguishedName, ATTRIBUTE, Attribute,
                  MATCHING-RULE FROM InformationFramework informationFramework
            CertificateSerialNumber, CertificateList, AlgorithmIdentifier,
                  EXTENSION
                  FROM AuthenticationFramework authenticationFramework
            DirectoryString
                  FROM SelectedAttributeTypes selectedAttributeTypes
            ub-name
                  FROM UpperBounds upperBounds
            ORAddress
                  FROM MTSAbstractService {joint-iso-ccitt mhs(6) mts(3)
                  modules(0) mts-abstract-service(1) version-1994 (0) } ;
-- Unless explicitly noted otherwise, there is no significance to the ordering
-- of components of a SEQUENCE OF construct in this Specification.
-- Key and policy information 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 ::= { ... }

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 }

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 }
--  Certificate subject and certificate issuer attributes extensions  --

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
--  Certification path constraints extensions  --

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)

CertPolicySet ::= SEQUENCE (1..MAX) OF CertPolicyId
--  Basic CRL extensions  --

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) }

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

HoldInstruction ::= OBJECT IDENTIFIER

invalidityDate EXTENSION ::= {
        SYNTAX          GeneralizedTime
        IDENTIFIED BY id-ce-invalidityDate }
--  CRL distribution points and delta-CRL extensions  --

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) }
```

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

IssuingDistPointSyntax ::= SEQUENCE {
        distributionPoint           [0] DistributionPointName OPTIONAL,
        onlyContainsUserCerts       [1] BOOLEAN DEFAULT FALSE,
        onlyContainsCACerts         [2] BOOLEAN DEFAULT FALSE,
        onlySomeReasons             [3] ReasonFlags OPTIONAL,
        indirectCRL                 [4] BOOLEAN DEFAULT FALSE }

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

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

BaseCRLNumber ::= CRLNumber

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

AltNameType ::= CHOICE {
        builtinNameForm     ENUMERATED {
                            rfc822Name              (1),
                            dNSName                 (2),
                            x400Address             (3),
                            directoryName           (4),
                            ediPartyName            (5),
                            uniformResourceIdentifier  (6),
                            iPAddress               (7),
                            registeredId            (8) },
        otherNameForm       OBJECT IDENTIFIER }

certificatePairExactMatch MATCHING-RULE      ::=          {
        SYNTAX          CertificatePairExactAssertion
        ID          id-mr-certificatePairExactMatch }

CertificatePairExactAssertion ::= SEQUENCE {
        forwardAssertion    [0] CertificateExactAssertion OPTIONAL,
        reverseAssertion    [1] CertificateExactAssertion OPTIONAL }
```

```
            ( WITH COMPONENTS         {..., forwardAssertion PRESENT} |
              WITH COMPONENTS         {..., reverseAssertion PRESENT} )

certificatePairMatch MATCHING-RULE ::=        {
            SYNTAX        CertificatePairAssertion
            ID            id-mr-certificatePairMatch }

CertificatePairAssertion ::= SEQUENCE {
            forwardAssertion    [0] CertificateAssertion OPTIONAL,
            reverseAssertion    [1] CertificateAssertion OPTIONAL }
            ( WITH COMPONENTS         {..., forwardAssertion PRESENT} |
              WITH COMPONENTS {..., reverseAssertion 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 }

algorithmIdentifierMatch MATCHING-RULE      ::=   {
            SYNTAX              AlgorithmIdentifier
            ID            id-mr-algorithmIdentifierMatch }
-- Object identifier assignments --

id-at-supportedAlgorithms           OBJECT IDENTIFIER   ::=   {id-at 52}
id-at-deltaRevocationList           OBJECT IDENTIFIER   ::=   {id-at 53}
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-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}
```

```
--  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
```

Annex B

Security requirements[2)]

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

Many OSI applications, ITU-T-defined services and non-ITU-T-defined services will have requirements for security. Such requirements derive from the need to protect the transfer of information from a range of potential threats.

B.1 Threats

Some commonly known threats are:

a) *Identity Interception* – The identity of one or more of the users involved in a communication is observed for misuse.

b) *Masquerade* – The pretense by a user to be a different user in order to gain access to information or to acquire additional privileges.

c) *Replay* – The recording and subsequent replay of a communication at some later date.

d) *Data Interception* – The observation of user data during a communication by an unauthorized user.

e) *Manipulation* – The replacement, insertion, deletion or misordering of user data during a communication by an unauthorized user.

f) *Repudiation* – The denial by a user of having participated in part or all of a communication.

g) *Denial of Service* – The prevention or interruption of a communication or the delay of time-critical operations

NOTE 1 – This security threat is a more general one and depends on the individual application or on the intention of the unauthorized disruption and is therefore not explicitly within the scope of the authentication framework.

h) *Mis-routing* – The mis-routing of a communication path intended for one user to another.

NOTE 2 – Mis-routing will naturally occur in OSI layers 1 through 3. Therefore mis-routing is outside of the scope of the authentication framework. However, it may be possible to avoid the consequences of mis-routing by using appropriate security services as provided within the authentication framework.

i) *Traffic Analysis* – The observation of information about a communication between users (e.g. absence/presence, frequency, direction, sequence, type, amount, etc.).

NOTE 3 – Traffic analysis threats are naturally not restricted to a certain OSI layer. Therefore Traffic analysis is generally outside the scope of the authentication framework. However, traffic analysis can be partially protected against by generating additional unintelligible traffic (traffic padding), using enciphered or random data.

B.2 Security services

In order to protect against perceived threats, various security services need to be provided. Security services as provided by the authentication framework are performed by means of the security mechanisms described in B.3:

a) *Peer entity authentication* – This service provides corroboration that a user in a certain instance of communication is the one claimed. Two different peer entity authentication services may be requested:

   – *single entity authentication* (either *data origin* entity authentication or *data recipient* entity authentication).

   – *mutual authentication*, where both users communicating authenticate each other.

   When requesting a peer entity authentication service, the two users agree whether their identities shall be protected or not.

   The peer entity authentication service is supported by the authentication framework. It can be used to protect against masquerade and replay, concerning the users' identities.

b) *Access control* – This service can be used to protect against the unauthorized use of resources. The access control service is provided by the Directory or another application and is therefore not a concern of the authentication framework.

[2)] For further information, see ISO 7498-2.

c) *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.

d) *Data integrity* – This service provides proof of the integrity of data in a communication. The data integrity service is supported by the authentication framework. It can be used to detect and protect against manipulation.

e) *Non-repudiation* – This service provides proof of the integrity and origin of data – both in an unforgeable relationship – which can be verifed by any third party at any time.

B.3 Security mechanisms

The security mechanisms outlined here perform the security services described in B.2.

a) *Authentication exchange* – There are two grades of authentication mechanism provided by the authentication framework:

- *Simple authentication* – Relies on the originator supplying its name and password, which are checked by the recipient.
- *Strong authentication* – Relies on the use of cryptographic techniques to protect the exchange of validating information. In the authentication framework, strong authentication is based upon an asymmetric scheme.

The authentication exchange mechanism is used to support the peer entity authentication service.

b) *Encipherment* – The authentication framework envisages the encipherment of data during transfer. Either asymmetric or symmetric schemes may be used. The necessary key exchange for either case is performed either within a preceding authentication exchange or off-line any time before the intended communication. The latter case is outside the scope of the authentication framework. The encipherment mechanism supports the data confidentiality service.

c) *Data integrity* – This mechanism involves the encipherment of a compressed string of the relevant data to be transferred. Together with the plain data, this message is sent to the recipient. The recipient repeats the compressing and subsequent encipherment of the plain data and compares the result with that created by the originator to prove integrity.

The data integrity mechanism can be provided by encipherment of the compressed plain data by either an asymmetric scheme or a symmetric scheme. (With the symmetric scheme, compression and encipherment of data might be processed simultaneously.) The mechanism is not explicitly provided by the authentication framework. However, it is fully provided as a part of the digital signature mechanism (see below) using an asymmetric scheme.

The data integrity mechanism supports the data integrity service. It also partially supports the non-repudiation service (that service also needs the digital signature mechanism for its requirements to be fully met).

d) *Digital signature* – This mechanism involves the encipherment, by the originator's private key, of a compressed string of the relevant data to be transferred. The digital signature together with the plain data is sent to the recipient. Similarly to the case of the data integrity mechanism, this message is processed by the recipient to prove integrity. The digital signature mechanism also proves the authenticity of the originator and the unambiguous relationship between the originator and the data that was transferred.

The authentication framework supports the digital signature mechanism using an asymmetric scheme.

The digital signature mechanism supports the data integrity service and also supports the non-repudiation service.

B.4 Threats protected against by the security services

Table B.1 indicates the security threats which each security service can protect against. The presence of a bullet ("•") indicates that a certain security service affords protection against a certain threat.

Table B.1 – Threats and protection

Threats	Services			
	Entity authentication	Data confidentiality	Data integrity	Non-repudiation
Identity interception	• (if required)			
Data interception		•		
Masquerade	•			
Replay	• (identity)		• (data)	•
Manipulation			•	
Repudiation				•

B.5 Negotiation of security services and mechanisms

The provision of security features during an instance of communication requires the negotiation of the context in which security services are required. This entails agreement on the type of security mechanisms and security parameters that are necessary to provide such security services. The procedures required for negotiating mechanisms and parameters can either be carried out as an integral part of the normal connection establishment procedure or as a separate process. The precise details of these procedures for negotiation are not specified in this annex.

Annex C

An introduction to public key cryptography[3)]

(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 C.1.

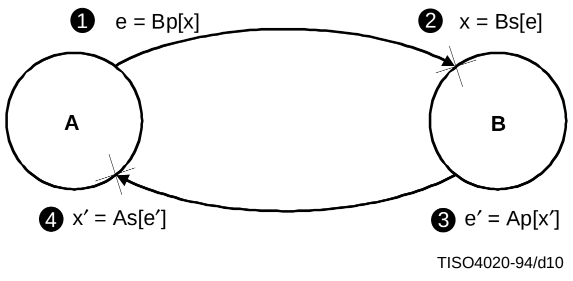


Figure C.1 – Use of a PKCS to exchange secret information

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 C.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′:

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

---

3) 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 D

The RSA[4] public key cryptosystem[5]

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

D.1 Scope and field of application

It is beyond the scope of this annex to discuss RSA fully. However, a brief description is given on the method, which relies on the use of modular exponentiation.

D.2 Definitions

The following terms are defined in this annex:

D.2.1 **public key**: The pair of parameters consisting of the Public Exponent and the Arithmetic Modulus.

NOTE – The ASN.1 data element **subjectPublicKey**, defined as **BIT STRING** (see Annex A), should be interpreted in the case of RSA as being of type:

SEQUENCE {INTEGER, INTEGER}

where the first integer is the Arithmetic Modulus and the second is the Public Exponent. The sequence is represented by means of the ASN.1 Basic Encoding Rules.

D.2.2 **private key**: The pair of parameters consisting of the Secret Exponent and the Arithmetic Modulus.

D.3 Symbols and abbreviations

For the purposes of this annex, the following symbols and abbreviations apply:

X,Y	Data blocks which are arithmetically less than the modulus
n	Arithmetic Modulus
e	Public Exponent
d	Secret Exponent
p,q	Prime numbers whose product forms the Arithmetic Modulus (n)
	NOTE – While the prime numbers are preferably two in number, the use of a Modulus with three – or more – prime factors is not precluded.
lcm	Least common multiple
mod n	Arithmetic modulo n

[4] The cryptosystem specified in this annex is widely known as *RSA*, Rivst-Shamir-Adleman.

[5] For further information, see:

General:

RIVEST (R. L.), SHAMIR, (A.) and ADLEMAN (L.): A Method for Obtaining Digital Signatures and Public-key Cryptosystems, *Communications of the ACM*, 21, 2, 120-126, February 1978.

Key Generation Reference:

GORDON (J.): Strong RSA Keys, *Electronics Letters*, 20, 5, 514-516.

Decipherment Reference:

QUISQUATER (J. J.) and COUVREUR (C.): Fast Decipherment Algorithm for RSA Public-key Cryptosystems, *Electronics Letters*, 18, 21, 905-907, October 14, 1982.

D.4 Description

This asymmetric algorithm uses the power function for transformation of data blocks such that:

- $Y = X^e \bmod n$ with $0 \le X < n$;
- $X = Y^d \bmod n$ with $0 \le Y < n$.

which may be satisfied, for example, by:

- $ed \bmod \mathrm{lcm}(p-1,q-1) = 1$; or
- $ed \bmod (p-1)(q-1) = 1$.

To effect this process, a data block shall be interpreted as an integer. This is accomplished by considering the entire data block to be an ordered sequence of bits (of length λ) where the first bit is the highest order bit of the first octet of the data block. The integer is then formed as the sum of the bits after giving a weight of $2^{\lambda-1}$ to the first bit and dividing by 2 for each subsequent bit (the last bit has a weight of 1).

The data block length should be the largest number of octets containing fewer bits than the modulus. Incomplete blocks should be padded in any way desired. Any number of blocks of additional padding may be added.

D.5 Security requirements

D.5.1 Key lengths

It is recognized that the acceptable key length is likely to change with time, subject to the cost and availability of hardware, the time taken, advances in techniques and the level of security required. It is recommended that a value for the length of n of 512 bits be adopted initially, but subject to *further study*.

D.5.2 Key generation

The security of RSA relies on the difficulty of factorizing n. There are many algorithms for performing this operation, and in order to thwart the use of any currently known technique, the values p and q shall be chosen carefully, according to the following rules (e.g. see footnote 5, "Key Generation Reference"):

a) They should be chosen randomly;

b) They should be large;

c) They should be prime;

d) $|p-q$ should be large;

e) $(p+1)$ shall possess a large prime factor;

f) $(q+1)$ shall possess a large prime factor;

g) $(p-1)$ shall possess a large prime factor, say r;

h) $(q-1)$ shall possess a large prime factor, say s;

i) $(r-1)$ shall possess a large prime factor;

j) $(s-1)$ shall possess a large prime factor.

After generating the public and private keys, e.g. "Xp" and "Xs" as defined in clauses 3 and 4, which consist of d, e and n, the values p and q together with all other data produced such as the product $(p-1)(q-1)$ and the large prime factors should preferably be destroyed. However, keeping p and q locally can improve throughput in decryption by two to four times. The decision to keep p and q is considered to be a local matter (see footnote 5 "Decipherment Reference").

It must be ensured that $e > \log_2(n)$. If not, then the simple operation of taking the integer *e*th root of a ciphertext block will disclose the plaintext.

D.6 Public exponent

The Public Exponent (e) could be common to the whole environment, in order to minimize the length of that part of the public key that actually has to be distributed, in order to reduce transmission capacity and complexity of transformation (see Note 1).

Exponent e should be large enough but such that exponentiation can be performed efficiently with regard to processing time and storage capacity. If a fixed public exponent e is desired, there are notable merits for the use of the Fermat Number F_4 (see Note 2).

$$F_4 = 2^{2^4} + 1$$

= 65537 decimal, and

= 1 0000 0000 0000 0001 binary

NOTE 1 – Although both Modulus n and Exponent e are public, the Modulus should not be the part which is common to a group of users. Knowledge of Modulus "n", Public Exponent "e", and Secret Exponent "d" is sufficient to determine the factorization of "n". Therefore, if the modulus were common, everyone could deduce its factors, thereby determining everyone else’s secret exponent.

NOTE 2 – The fixed exponent should be large and prime but it should also provide efficient processing. Fermat Number F_4 meets these requirements, e.g. authentication takes only 17 multiplications and is on the average 30 times faster than decipherment.

D.7 Conformance

Whilst this annex specifies an algorithm for the public and secret functions, it does not define the method whereby the calculations are carried out; therefore, there may be different products which comply with this annex and are mutually compatible.

Annex E

Hash functions

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

The square-mod hash function that was described in this annex in the first edition of this Directory Specification is deprecated.

E.1 Requirements for hash functions

To use a hash function as a secure one-way function, it shall not be possible to obtain easily the same hash result from different combinations of the input message.

A strong hash function shall meet the following requirements:

a) The hash function shall be one-way, i.e. given any possible hash result, it shall be computationally infeasible to construct an input message which hashes to this result.

b) The hash function shall be collision-free, i.e. it shall be computationally infeasible to construct two distinct input messages which hash to the same result.

Annex F

Threats protected against by the strong authentication method

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

The strong authentication method described in this Directory Specification offers protection against the threats as described in Annex B for strong authentication.

In addition, there is a range of potential threats that are specific to the strong authentication method itself. These are:

– *Compromise of the user's private key* – One of the basic principles of strong authentication is that the user's private key remain secure. A number of practical methods are available for the user to hold his private key in a manner that provides adequate security. The consequences of the compromise is limited to subversion of communication involving that user.

– *Compromise of the CA's private key* – That the private key of a CA remain secure is also a basic principle of strong authentication. Physical security and "need to know" methods apply. The consequences of the compromise are limited to subversion of communication involving any user certified by that CA.

– *Misleading CA into producing an invalid certificate* – The fact that CAs are off-line affords some protection. The onus is on the CA to check that purported strong credentials are valid before creating a certificate. The consequences of the compromise are limited to subversion of communication involving the user for whom the certificate was created, and anyone impacted by the invalid certificate.

– *Collusion between a rogue CA and user* – Such a collusive attack will defeat the method. This would constitute a betrayal of the trust placed in the CA. The consequences of a rogue CA are limited to subversion of communication involving any user certified by that CA.

– *Forging of a certificate* – The strong authentication method protects against the forging of a certificate by having the CA sign it. The method depends on maintaining the secrecy of the CA's private key.

– *Forging of a token* – The strong authentication method protects against the forging of a token by having the sender sign it. The method depends on maintaining the secrecy of the sender's private key.

– *Replay of a token* – The one- and two-way authentication methods protect against the replay of a token by the inclusion of a timestamp in the token. The three-way method does so by checking the random numbers.

– *Attack on the cryptographic system* – The likelihood of effective cryptanalysis of the system, based on advances in computational number theory and leading to the need for a greater key length are reasonably predictable.

Annex G

Data confidentiality

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

G.1 Introduction

The process of data confidentiality can be initiated after the necessary keys for encipherment have been exchanged. This might be provided by a preceding authentication exchange as described in clause 9 or by some other key exchange process, the latter being outside the scope of this Directory Specification.

Data confidentiality can be provided either by the application of an asymmetric or symmetric enciphering scheme.

G.2 Data confidentiality by asymmetric encipherment

In this case, Data Confidentiality is performed by means of an originator enciphering the data to be sent using the intended recipient's public key: the recipient shall then decipher it using its private key.

G.3 Data confidentiality by symmetric encipherment

In this case, Data Confidentiality is achieved by the use of a symmetric enciphering algorithm. Its choice is outside the scope of the authentication framework.

Where an authentication exchange according to clause 9 has been carried out by the two parties involved, then a key for the usage of a symmetric algorithm can be derived. Choosing private keys depends on the transformation to be used. The parties shall be sure that they are strong keys. This Directory Specification does not specify how this choice is made, although clearly this would need to be agreed by the parties concerned, or specified in other standards.

Annex H

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-ccitt ds(5) module(1) algorithmObjectIdentifiers(8)3}
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-ccitt ds(5) module(1) usefulDefinitions(0) 3}
        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 I

Bibliography

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

KENT (S.): Privacy Enhancement for Internet Electronic Mail, Part II: Certificate-Based Key Management, RFC 1422, *Internet Activities Board*, 1993.

United Nations, EDIFACT Security Implementation Guidelines, UN *Economic and Social Council*, TRADE/WP.4/R.1026, 7 February 1994. (To be incorporated in ISO 9735 EDIFACT Syntax.)

Annex J

Examples of use of certification path constraints

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

J.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 }

J.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}}}**

J.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 K

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;

– Amendment 2 for Enhancement of Directory Operational Security.

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 Report 128);

– Technical Corrigendum 2 (covering Defect Reports 077, 078, 083, 084);

– Technical Corrigendum 3 (covering Defect Reports 080, 092, 100, 177);

– Technical Corrigendum 1 to the 97 edition (covering Defect Reports 183, 194).