INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.843

(10/2000)

SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

Security

Information technology – Security techniques – Specification of TTP services to support the application of digital signatures

ITU-T Recommendation X.843

(Formerly CCITT Recommendation)

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

INTERNATIONAL STANDARD ISO/IEC 15945

ITU-T RECOMMENDATION X.843

INFORMATION TECHNOLOGY – SECURITY TECHNIQUES – SPECIFICATION OF TTP SERVICES TO SUPPORT THE APPLICATION OF DIGITAL SIGNATURES

Summary

This Recommendation | International Standard defines the services required to support the application of digital signatures for non-repudiation of creation of a document. Since this implies integrity of the document and authenticity of the creator, the services described can also be combined to implement integrity and authenticity services.

Source

ITU-T Recommendation X.843 was prepared by ITU-T Study Group 7 (1997-2000) and approved by the World Telecommunication Standardization Assembly (Montreal, 27 September - 6 October 2000). An identical text is also published as ISO/IEC 15945.

CONTENTS

Introduction

Today the development of information technology, as well as that of the worldwide communication infrastructure, opens the possibility to implement electronic commerce in economically relevant dimensions. Digital signatures are an important technique to add security to these commercial applications and to other application fields with a need for legally effective electronic transactions.

Digital signatures are suitable to assure the integrity of data and the authentication of participants in transactions. They can supply an analogue of the handwritten signature for digital orders, offers and contracts. The most important property of digital signatures in this context is that a person who signed a document cannot successfully deny this fact. This property is called "non-repudiation of creation" of a document.

In several countries and in international contexts, legislation concerning digital signatures is being pushed forward with the aim to support the development of electronic commerce and other application fields with a need for legally effective electronic transactions.

A number of standards exist that specify digital signatures, as well as their use for different purposes, like non-repudiation or authentication. A number of commercial applications, as well as TTPs offering services in connection with digital signatures, are implemented or planned. Interoperability of these TTPs, among each other and with the commercial applications, is needed for an economically and legally effective worldwide use of digital signatures.

The goal of this Recommendation | International Standard is to define the services required to support the application of digital signatures for non-repudiation of creation. Since the use of digital signature mechanisms for non-repudiation of creation of a document implies integrity of the document and authenticity of the creator, the services described in this Recommendation | International Standard can also be combined to implement integrity and authenticity services. This is done in a way to promote interoperability among TTPs as well as between TTPs and commercial applications.

NOTE – There is no inherent reason why every TTP planning to support the application of digital signatures should be required to offer all of these services. It is possible that a number of TTPs offering different services cooperate in supporting the use of digital signatures. But, from the view of the potential commercial applications, the whole range of the services may be required and interoperability becomes even more important in this scenario. This is an additional justification to collect all these services together in one document.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – SECURITY TECHNIQUES – SPECIFICATION OF TTP SERVICES TO SUPPORT THE APPLICATION OF DIGITAL SIGNATURES

1 Scope

This Recommendation | International Standard will define those TTP services needed to support the application of digital signatures for the purpose of non-repudiation of creation of documents.

This Recommendation | International Standard will also define interfaces and protocols to enable interoperability between entities associated with these TTP services.

Definitions of technical services and protocols are required to allow for the implementation of TTP services and related commercial applications.

This Recommendation | International Standard focuses on:

– implementation and interoperability;

– service specifications; and

– technical requirements.

This Recommendation | International Standard does not describe the management of TTPs or other organizational, operational or personal issues. Those topics are mainly covered in ITU-T Rec. X.842 | ISO/IEC TR 14516, *Information technology – Security techniques – Guidelines on the use and management of Trusted Third Party services*.

NOTE 1 – Because interoperability is the main issue of this Recommendation | International Standard, the following restrictions hold:

i) Only those services which may be offered by a TTP, either to end entities or to another TTP, are covered in this Recommendation | International Standard.

ii) Only those services which may be requested and/or delivered by means of standardizable digital messages are covered.

iii) Only those services for which widely acceptable standardized messages can be agreed upon at the time this Recommendation | International Standard is published are specified in detail.

Further services will be specified in separate documents when widely acceptable standardized messages are available for them. In particular, time stamping services will be defined in a separate document.

NOTE 2 – The data structures and messages in this Recommendation | International Standard will be specified in accordance to RFC documents, RFC 2510 and RFC 2511 (for certificate management services) and to RFC 2560 (for OCSP services). The certificate request format also allows interoperability with PKCS#10. See Annex C for references to the documents mentioned in this Note.

NOTE 3 – Other standardization efforts for TTP services in specific environments and applications, like SET or EDIFACT, exist. These are outside of the scope of this Recommendation | International Standard.

NOTE 4 – This Recommendation | International Standard defines technical specifications for services. These specifications are independent of policies, specific legal regulations, and organizational models (which, for example, might define how duties and responsibilities are shared between Certification Authorities and Registration Authorities). Of course, the policy of TTPs offering the services described in this Recommendation | International Standard will need to specify how legal regulations and the other aspects mentioned before will be fulfilled by the TTP. In particular, the policy has to specify how the validity of digital signatures and certificates is determined.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.501 (1997) | ISO/IEC 9594-2:1998, *Information technology – Open Systems Interconnection – The Directory: Models*.
- ITU-T Recommendation X.509 (2000) | ISO/IEC 9594-8:2001, *Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks*.
- ITU-T Recommendation X.520 (1997) | ISO/IEC 9594-6:1998, *Information technology – Open Systems Interconnection – The Directory: Selected attribute types*.
- ITU-T Recommendation X.680 (1997) | ISO/IEC 8824-1:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.
- ITU-T Recommendation X.681 (1997) | ISO/IEC 8824-2:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.
- ITU-T Recommendation X.682 (1997) | ISO/IEC 8824-3:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification*.
- ITU-T Recommendation X.683 (1997) | ISO/IEC 8824-4:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.
- ITU-T Recommendation X.690 (1997) | ISO/IEC 8825-1:1998, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*.
- ITU-T Recommendation X.810 (1995) | ISO/IEC 10181-1:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Overview*.
- ITU-T Recommendation X.813 (1996) | ISO/IEC 10181-4:1997, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Non-repudiation framework*.

2.2 Additional references

- ISO/IEC 9796-2:1997, *Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Mechanisms using a hash-function*.
- ISO/IEC 9796-3:2000, *Information technology – Security techniques – Digital signature schemes giving message recovery – Part 3: Discrete logarithm based mechanisms*.
- ISO/IEC 10118-1:1994, *Information technology – Security techniques – Hash-functions – Part 1: General*.
- ISO/IEC 10118-2:1994, *Information technology – Security techniques – Hash-functions – Part 2: Hash-functions using an n-bit block cipher algorithm*.
- ISO/IEC 10118-3:1998, *Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions*.
- ISO/IEC 11770-1:1996, *Information technology – Security techniques – Key management – Part 1: Framework*.
- ISO/IEC 11770-2:1996, *Information technology – Security techniques – Key management – Part 2: Mechanisms using symmetric techniques*.
- ISO/IEC 11770-3:1999, *Information technology – Security techniques – Key management – Part 3: Mechanisms using asymmetric techniques*.
- ISO/IEC 13888-1:1997, *Information technology – Security techniques – Non-repudiation – Part 1: General*.
- ISO/IEC 13888-2:1998, *Information technology – Security techniques – Non-repudiation – Part 2: Mechanisms using symmetric techniques*.
- ISO/IEC 13888-3:1997, *Information technology – Security techniques – Non-repudiation – Part 3: Mechanisms using asymmetric techniques*.
- ISO/IEC 14888-1:1998, *Information technology – Security techniques – Digital signatures with appendix – Part 1: General*.

- ISO/IEC 14888-2:1999, *Information technology – Security techniques – Digital signatures with appendix – Part 2: Identity-based mechanisms*.
- ISO/IEC 14888-3:1998, *Information technology – Security techniques – Digital signatures with appendix – Part 3: Certificate-based mechanisms*.
- ISO/IEC 15946-2 (to be published), *Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 2: Digital signatures*.

3 Definitions

The following term is defined in ISO/IEC 11770-1:

key management

The following term is defined in ISO/IEC 10181-1:

Trusted Third Party (TTP)

The following terms are defined in ITU-T Rec. X.509 | ISO/IEC 9594-8:

CA-certificate
Certification Authority (CA)
public key certificate

NOTE – The shorter term "certificate" will also be used in this Recommendation | International Standard to denote "public key certificate".

certificate policy
certificate revocation list (CRL)
certification path

The following terms are defined in ISO/IEC 10118-1:

hash function
hash-code (hash-value)

The following term is defined in ISO/IEC 14888-1:

domain parameter

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

3.1 **certificate management services**: All services needed for the maintenance of the lifecycle of certificates, including registration, certification, distribution, and revocation of certificates.

3.2 **certification service**: The service of creating and assigning certificates performed by a CA and described in ISO/IEC 9594-8:1995.

3.3 **digital signature:** A cryptographic transformation of a data unit that allows a recipient of the data unit to prove the origin and integrity of the data unit and protect the sender and the recipient of the data unit against forgery by third parties, and the sender against forgery by the recipient.

NOTE – Digital signatures may be used by end entities (see below) for the purposes of authentication, of data integrity, and of non-repudiation of creation of data. The usage for non-repudiation of creation of data is the most important one for legally binding digital signatures. The definition above is taken from ISO/IEC 9798-1.

3.4 **directly trusted CA**: A directly trusted CA is a CA whose public key has been obtained and is being stored by an end entity in a secure, trusted manner, and whose public key is accepted by that end entity in the context of one or more applications.

3.5 directly trusted CA key: A directly trusted CA key is a public key of a directly trusted CA. It has been obtained and is being stored by an end entity in a secure, trusted manner. It is used to verify certificates without being itself verified by means of a certificate created by another CA.

NOTE – If, for example, the CAs of several organizations cross-certify each other (see Annex A), the directly trusted CA for an entity may be the CA of the entity's organization. Directly trusted CAs and directly trusted CA keys may vary from entity to entity. An entity may regard several CAs as directly trusted CAs.

3.6 directory service: A service to search and retrieve information from a catalogue of well defined objects, which may contain information about certificates, telephone numbers, access conditions, addresses, etc. An example is provided by a directory service conforming to the ITU-T Rec. X.500 | ISO/IEC 9594-1.

3.7 key distribution service: The service of distributing keys securely to authorized entities performed by a Key Distribution Center and described in ISO/IEC 11770-1.

3.8 non-repudiation of creation: Protection against an entity's false denial of having created the content of a message (i.e. being responsible for the content of a message).

3.9 personal security environment (PSE): Secure local storage for an entity's private key, the directly trusted CA key and possibly other data. Depending on the security policy of the entity or the system requirements, this may be, for example, a cryptographically protected file or a tamper resistant hardware token.

3.10 personalization service: The service of storing cryptographic information (especially private keys) to a PSE.

NOTE – The organizational and physical security measures for a service like this are not in the scope of this Recommendation | International Standard. For organizational measures, refer to ITU-T Rec. X.842 | ISO/IEC TR 14516 Guidelines on the use and management of Trusted Third Party services.

3.11 public key directory (PKD): A directory containing a well defined (sub)set of public key certificates. This directory can contain certificates from different Certification Authorities.

3.12 public key infrastructure (PKI): The system consisting of TTPs, together with the services they make available to support the application (including generation and validation) of digital signatures, and of the persons or technical components, who use these services.

NOTE – Sometimes the persons and the technical components participating in a PKI, by using the services of TTPs, but not being TTPs themselves, are referred to as end entities. An example of a technical equipment used by an end entity is a smartcard which may be used as a storage and/or processing device.

3.13 registration authority (RA): Authority entitled and trusted to perform the registration service as described below.

3.14 registration service: The service of identifying entities and registering them in a way that allows the secure assignment of certificates to these entities.

3.15 time stamping service: A service which attests the existence of electronic data at a precise instant of time.

NOTE – Time stamping services are useful and probably indispensable to support long-term validation of signatures. They will be defined in a separate document.

4 Abbreviations

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

CA	Certification Authority
CRL	Certificate Revocation List
EE	End Entity
OCSP	On-line Certificate Status Protocol
PKD	Public Key Directory
PKI	Public Key Infrastructure
PSE	Personal Security Environment
RA	Registration Authority
TTP	Trusted Third Party

5 Descriptive classification of services

This clause describes services that may be used within the context of TTP services for digital signatures. Here a high-level description for those services is given, this description is independent of data formats, algorithms, description languages, etc.

A detailed specification of some of these services is given in the following clauses.

It is not required that every TTP planning to support the application of digital signatures offers all of these services. It is possible that a number of TTPs, offering different services, cooperate in supporting the use of digital signatures.

NOTE 1 – Because interoperability is the main issue of this Recommendation | International Standard, only those services which are offered by a TTP, either to end entities or to another TTP, are described here. Furthermore, only those services are covered which may be requested and/or delivered by means of standardizable digital messages. (However, this does not imply that standardized messages are in fact defined for all services mentioned in this Recommendation | International Standard.)

The following examples show services that are **not** covered:

1) Logging of security relevant events. With respect to a digital signature PKI this is an internal service of TTPs but not offered to entities.
2) General cryptographic services (e.g. encryption service). Processes like encryption are part of some services but not relevant as stand-alone service in the context of digital signatures.
3) Key archiving or recovery. This may be an internal service for directly trusted CA keys. This will usually not be done for digital signature keys of end entities.

NOTE 2 – Time stamping services will be defined in a separate document.

5.1 Certificate management services

This subclause contains a description of the following services that are part of the certificate lifecycle:

- Registration;
- Public Key Certification;
- Revocation of Certificates;
- Certificate Update; and
- Key Update.

A detailed specification for the online message flow of these services (except certificate status determination) is given in clause 7, and the ASN.1 specification of the data structures needed for these messages is given in clause 8. The analogue specifications for online certificate status determination (see 5.1.3.3, second method) are given in clause 9.

The directory access protocols used to make certificates and CRLs publicly available are not specified here since specifications for these protocols already exist in ITU-T Rec. X.511 | ISO/IEC 9594-3 and ITU-T Rec. X.519 | ISO/IEC 9594-5.

NOTE – Other protocols for directory access are LDAP (RFCs 1777, 2555 and 2587-LDAPv2) or WEB-access (RFC 2585).

Figure 1 gives an overview over the architecture of a PKI with some example services.

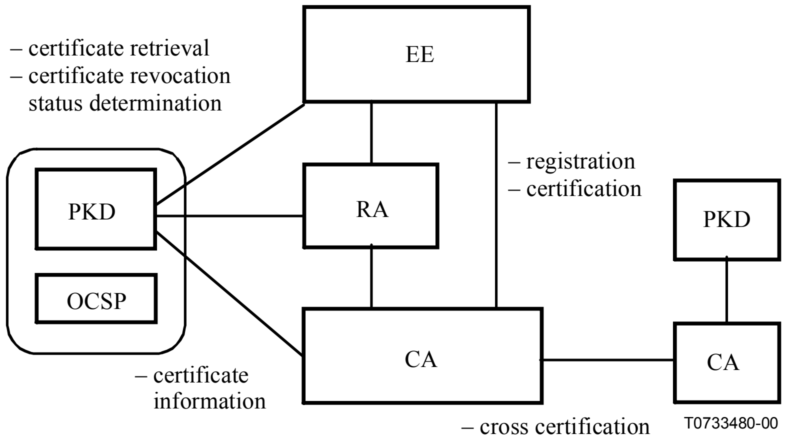

Figure 1 – Overview of Certificate Management Services

5.1.1 Registration

The trustworthiness of any Public Key infrastructure relies on the proper identification and registration of entities.

For all end entities, an RA (which may also be the CA) has to verify the end entity's identity by suitable means, and has to assign an unambiguous and unique name to each end entity within its own domain. The certificate policy determines the kind of means to be used for identification (e.g. ID documents) and whether the end entity has to be present in person. Aliasing of end entity's names may also be required. Those end entities which are technical components are registered according to the security policy for the application, e.g. network management. Applications, for which the distinction between human and non-human end entities is important, should make this distinction clear within the certificate. For example, a "name" could be "device type X", number "Y" located in "Z", the distinction could be made according to policy, or an extension may indicate the difference.

A registration form may be used to collect the relevant data, e.g. name, business unit, business address, and delivery address for the keying material.

EXAMPLE: A scenario within a company may be that each employee has to be present in person at the relevant personnel office, showing his valid identity card. The personnel officer in charge attests the identity and sends a signed registration form to the CA.

5.1.2 Public Key certification

The CA is responsible for the certification process, the binding of an entity's name or a pseudonym with public keys. A certificate format is described in ITU-T Rec. X.509 | ISO/IEC 9594-8.

This Recommendation | International Standard requires that proof of possession of the private key corresponding to the public key included in the certificate is carried out in the course of the certification service. Depending on policy, a CA may guarantee further properties of the public key of the end entity, e.g. by including one or both of the services described in 5.3.2 and 5.3.3.

5.1.3 Revocation of certificates

5.1.3.1 General considerations

A major concern with public key certificates is that they may have to be *revoked* prior to their scheduled expiration time due to different circumstances. Revoking may be done by the issuing CA or by another authority, depending on the policy.

The revocation is mandatory, if a private key is compromised. Other reasons may be change of name, termination of employment, etc.

The certificate policy should state all revocation reasons. Some of these reasons can be found in ITU-T Rec. X.509 | ISO/IEC 9594-8. The policy should include who can request revocations, and also if a specific token can be used to identify entities who are authorized to revoke certificates such as one-time revocation passwords.

5.1.3.2 Revocation methods

Two different methods can be used to revoke certificates:

1) Issuing of a Certificate revocation list (CRL).

   A CRL is a periodically issued list which is signed by the CA and identifies all certificates that are revoked.

   Revocation leads to an immediate entry in the CRL which also includes the time of revocation of the concerned certificate. Additionally, the policy of the CA may require that the certificate is removed from the PKD.

   NOTE – Depending on the policy, it may be suitable to keep the certificate in order to check the validity of signatures made before the revocation date (e.g. if the revocation reason is not key compromise).

   In addition to the periodic publishing, the updated CRL may be published immediately.

   ISO/IEC 11770-1 identifies two different revocation time stamps:

   – The time of known or suspected compromise; and

   – the time at which the CA was notified of the compromise by the entity.

   Depending on the policy, the entry of the certificate in the CRL may be deleted when the certificate expires (compare 5.1.3.3 Certificate revocation status determination).

2) Storing the status of the certificate in an internal trusted database of the CA and offering online certificate status information to entities (compare 5.1.3.3 Certificate revocation status determination).

Both methods may be combined.

If a key is compromised, the standard procedure is to revoke the corresponding certificate, re-initialize the end entity, generate a new public/private key pair, and certify the new public key with the previous attributes.

Depending on the policy, a certificate may:

- be revoked permanently;
- be suspended. The CRL entry includes a flag indicating the status "on hold".

The policy of the CA shall specify what the status "on hold" means with regard to the level of trust it represents to an entity, and how an entity should treat this situation. For example, a certificate might be suspended as a result of an unauthorized revocation request. Some policies do not allow the status "on hold" at all.

EXAMPLE: A policy may state the following: Whenever a verification is performed, it shall be rejected as long as the certificate is suspended. When an evidence is verifiable using a suspended certificate, the result of the verification shall be negative if a result is immediately needed, but can also be interpreted as a conditional validity. When the suspension is followed by a revocation, then the evidence becomes invalid and the revocation date shall be the date of the start of the suspension (i.e. not the date of the end of the suspension).

5.1.3.3 Certificate revocation status determination

This service allows an entity to determine if a certificate is revoked. This can be done by different methods corresponding to the methods of revocation as described in 5.1.3.2:

1) Method 1: Checking the PKD and CRL.
2) Method 2: To request on-line the status from a TTP that is trusted for that purpose. The TTP's answer has to be conveyed in an authentic manner to the entity.

5.1.3.4 Revocation of a CA certificate

A special scenario is encountered if a CA's private key is compromised. If that occurs:

- the certificate of the public key corresponding to the CA's compromised key has to be revoked.

  Trust in the certificates, calculated with the compromised key of the CA, is not provided any more by the CA's signature contained in these certificates. However, it is possible to guarantee the validity of such certificates with other means (e.g. if the certificates are stored in a trustworthy way by a TTP which provides an OCSP service). In any case, the entities should obtain a new certificate and a new directly trusted CA key for future signatures.

  Depending on the policy, the certificates calculated with the compromised key are revoked as a means of informing the end entities about the necessity of getting new certificates.

  In some situations, depending on the certificate policy and the system architecture, new key pairs should be generated for the entities.

  It is important to note that under this scenario, any signatures that were issued before the compromise occurred, and are made secure by additional means (e.g. that are time-stamped by a time stamping authority whose certificate is still valid), may still be considered valid, depending on policy, while any signatures issued after the time of determined compromise, or not made secure by additional means, will be considered invalid.
- if the public key corresponding to the CA's compromised private key is cross-certified with other CAs, an alert message has to be sent to the other CAs. This alert message will notify them to revoke the cross-certified CAs certificate.

5.1.4 Certificate update

In case of expiration of a certificate, it may be updated by issuing a new certificate for the public key of the entity that was already contained in the old certificate.

This method shall not be used if:

- the key pair of the entity is compromised; or
- the state of cryptography indicates that the public key algorithm in connection with the parameters of the key pair may not guarantee security of signatures generated with this key pair for the validity period of the new certificate; or
- the new certificate will have substantial differences in terms of policy, extensions or attributes from the old certificate.

The certificate policy of the CA may specify that a simplified registration procedure is acceptable to issue a new certificate.

EXAMPLE: If the old certificate is not revoked, this may be accepted as sufficient to issue a new certificate.

The change of non-critical attributes in a certificate during its period of validity, such as name or affiliation (e.g. by a change to another department), may also lead to the requirement for a recertification of the amended attributes with the same keys as before. Nevertheless, in this case the previous certificate has to be revoked.

5.1.5 Key update

A new key pair is generated, either by the entity itself or by the TTP, and a certificate is issued for the public key of this new pair.

This method shall be chosen in case of expiration of a certificate if certificate update is not acceptable for one of the reasons given in 5.1.4. Depending on the policy of the CA, it may also be used in other situations.

The certificate policy of the CA may specify that a simplified registration procedure is acceptable to issue a certificate for an updated key.

5.2 Key management services

General descriptions for key management services are given in ISO/IEC 11770 (all parts).

This subclause contains a description only of those key management services that may be offered as a part of services related to the certificate lifecycle (compare 5.1). The detailed specification for the on-line message flow of those services in clause 7, and the ASN.1 specification of the data structures in clause 8, cover the key management services as far as they result in on-line messages between CA, RA and/or EE.

5.2.1 Key generation

In the context of digital signatures, TTPs may generate private/public key pairs for end entities if this is not done by the entities themselves. Though the service might be offered by independent TTPs, it is assumed further on that it is done either by a CA or a RA in response to a certification request or by the end entity prior to a certification request.

5.2.2 Key distribution

5.2.2.1 Distribution of private keys

For the description of the service, "key distribution", different modes of key transmission (on-line or off-line) and the key generating component (TTP or entity) can be distinguished. In the case of a centralized key generation, the TTP is responsible for a secure transmission of the entity's private key and public key certificate. Moreover, it must be ensured that an entity's private key will be sent in a confidential manner. This can be achieved by encrypting this key with a special (symmetric) transportation key known only to the TTP and the corresponding entity. Alternatively, the private key may be transmitted using appropriate secure hardware facilities as for example, smartcards. The transmission of the private key is not necessary if the entity is able to generate its own asymmetric key pair. In this case, the TTP merely has to perform some plausibility checks (e.g. if the entity is able to sign a message with a private key corresponding to the public key), to certify the entity's public key, and to make this certificate available.

5.2.2.2 Distribution of public keys

Public keys must be made available to entities in a way that guarantees their authenticity. In the case of certified public keys, the key distribution is done by distribution of the certificate and authenticity is guaranteed by the signature of the CA which has created the certificate.

In the case of a directly trusted CA key, other means of secure distribution have to be used. If private keys of an entity are distributed to an entity using a secure hardware token, this token can also be used for delivery of the CA key. In other cases an additional process is required. See ISO/IEC 11770-3, subclause 8.1 "Public key distribution without a trusted third party" for methods to do this.

5.2.3 Personalization

The storage of private keys and additional data may be provided by using a physical token. In this situation the personalization of a token has to be supported by the CA, RA or end entities. For example, the personalization of smartcards may include set-up procedures (e.g. creation of the file system), the selection of a random PIN (Personal Identification Number) or password and the shipment and storage of all relevant data within a smartcard.

5.3 Other services

5.3.1 Cross certification

Cross certification is a service offered to allow verification of signatures from end entities with certificates from one CA by end entities with certificates from another CA. For example, a CA1 issues a certificate for a CA2 in another PKI with the effect that entities trusting CA1 can verify certificates of entities in the other PKI via a certification path including this new certificate.

A detailed specification for the on-line message flow of this service is given in clause 7 and the ASN.1 specification of the data structures needed for these messages is given in clause 8.

5.3.2 Domain parameter validation

Domain Parameter Validation is the validation of a proposed set of domain parameters to ensure that each parameter in the set meets all the attributes that are claimed for it.

EXAMPLES:

a) a valid parameter may be required to be a prime: to validate this, a primality test (perhaps probabilistic) is run to make sure that the claimed prime is actually a prime;

b) a valid parameter may be required to be in some arithmetic relationship with some other parameter(s): to validate this, the arithmetic relationship is tested to ensure it holds;

c) a specific weak case (for example, on an exclude list) may be checked to ensure it does not apply to the set in question; or

d) a parameter may be required to be generated by use of a seed in a canonical seeded hash function: to validate this, the seed is input to the canonical seeded hash function to ensure that it actually does generate the parameter.

The generator of a set of domain parameters should ensure that they pass domain parameter validation. Whether anyone else needs to do domain parameter validation depends on the trust relationship between the generator and the entity. If a set of invalid domain parameters is used, unpredictable results may occur, including loss of any intended security. As domain parameters are typically public, it is best if the validation can be done in an off-line manner (that is, not needing the generator of the domain parameters to answer queries) and this is typically the case.

Usually a CA will generate and validate a set of domain parameters which can then be implicitly trusted by all members of the PKI.

5.3.3 Public key validation

Public Key Validation is the validation of a claimed public key to ensure it conforms to the arithmetic requirements for such a key, that is, that the claimed public key is plausible. Public Key Validation assumes that any domain parameters have previously been validated.

EXAMPLES:

a) a valid parameter may be required to be in a specific range of values: to validate this, the claimed parameter is tested to ensure it is in the correct range;

b) a valid parameter may be required to have a specific order (typically a large prime order): to validate this, the claimed parameter is tested to ensure it has the correct order; or

c) a valid parameter may be required to be in a specific arithmetic relationship with some other parameter(s), to validate this, the arithmetic relationship is tested.

If an invalid public key is used, unpredictable results may occur, including loss of any intended security of the owner of the associated private key, recipients of documents signed with that key, or both. A trusted third party, such as a CA, can do Public Key Validation to assure all entities in its domain.

5.3.4 Certificate validation

If an end entity that wants to rely on a digital signature of another end entity is not able to verify the corresponding certificate, it may ask a TTP to do this.

Certificate validation concerns the validity of a single certificate. A single certificate may be supplied in the request, or that single certificate may be supplied and followed by a sequence of certificates (not necessarily forming a certification path).

There are two conditions to be met in order to test the validity of a certificate:

a) A single certificate may only be valid against a certificate policy. It is thus necessary to know which certificate policy applies. The certificate policy defines, among other things, a rule, how to build a valid certification path. So the task is to determine if it is possible to build a certification path. If the supplied certificates are not sufficient, the service may attempt to gather itself the missing certificates. There can be different ways to identify the certificate policy, but there must be some pointer to point to the right policy. For this, an OID (Object Identifier) or a URL can be used. In order to make sure that the pointer is correct, a hash code value of the policy should be added. A simple and degenerated form for a policy is a single self-signed certificate. In that case, it is needed to point to this self-signed certificate by providing the name of the CA and the serial number of the certificate.

b) It is also important to know on which date the validity test should be made. This is of particular importance in order to make sure that the certificate has not been revoked prior to that date. Depending on the certificate policy, the validity not only of the certificate itself but also of all certificates in the certificate path will have to be checked for points in time, which may be different from each other and may which depend on the signature time. The knowledge of the signature date is thus of particular importance.

As a conclusion, the input parameters for this service would be:

– an identifier of the certificate to be tested for validity, followed by zero or more certificates;

– zero, one or more CRLs (Certificate Revocation Lists);

– the identifier of a certificate policy against which the certificate shall be tested, i.e. a pointer such as an OID or a URL followed by a hash of the certificate policy, or the identifier of a self-signed certificate (in a degenerated case);

– the instant of time for which the test shall be made.

The output parameter should be enumerated.

5.3.5 Archival service

Since digital signatures may be used to sign documents that have to be valid for a long time, it must be possible to determine the validity of a signature even a long time after the expiration of the corresponding certificate. Special provisions exceeding the normal CA procedures have to be made for this, since the certificate validity period is the time interval during which the CA normally warrants that it will maintain information about the status of the certificate. The policy of a conforming PKI shall define these provisions. Possible provisions are the use of time-stamping services or archival services.

Archival services may be important to resolve conflicts concerning digital signatures. For example, certificates, CRLs and other data may be archived by dedicated TTPs, since usual directory services may only contain certificates and CRLs that are not expired. To achieve this, the history of the lifetime of certificates and CRLs may be recorded in such a way that their validity time frame can be reconstructed. The certificate policy may require that the CA itself archives all certificates and CRLs that have ever been issued.

6 Minimal certificate and CRL profile

6.1 Minimal certificate profile

Certificates shall comply to X.509 format as specified in ITU-T Rec. X.509 | ISO/IEC 9594-8. Entities verifying certificates shall be able to process this format.

No full certificate profile will be given here, but the following certificate fields shall be implemented for reasons of security and/or interoperability (the names of the certificate fields refer to ITU-T Rec. X.509 | ISO/IEC 9594-8):

a) The version shall at least be v3.

b) The following extensions shall be used by CAs and applications shall be able to process them:

   – basic constraints;

   – key usage;

   – authority key identifier (possible exception: self-signed certificates for directly trusted CA keys);

   – certificate policies.

c) Applications shall be able to process the subject alternative name extension. CAs shall use this extension if the subject field of a certificate is empty.

d) Applications shall be able to process the name constraints, policy constraints and extended key usage extensions. The name constraints extension shall be used in CA certificates only.

6.2 Minimal CRL profile

CRLs shall comply to X.509 format as specified in ITU-T Rec. X.509 | ISO/IEC 9594-8. Entities using CRLs shall be able to process this format.

No full CRL profile will be given here, but the following fields shall be implemented for reasons of security and/or interoperability (the names of the certificate fields refer to ITU-T Rec. X.509 | ISO/IEC 9594-8):

a) The CRL-version shall be at least v2 as defined in ITU-T Rec. X.509 | ISO/IEC 9594-8.

b) CRLs shall contain the next update field, the CRL number extension and the keyIdentifier field of the authority key identifier extension. Applications shall be able to process these fields.

7 Certificate management messages

Messages are defined in this clause for all relevant services of certificate management. The data structures will be specified in accordance with RFC documents RFC 2510 and RFC 2511. The messages will be specified in detail in clause 8 using ASN.1. The ASN.1 syntax is defined in ITU-T Recs. X.680-X.683 | ISO/IEC 8824 parts 1-4, coding rules are described in ITU-T Rec. X.690 | ISO/IEC 8825-1.

NOTE 1 – Note that on-line protocols are not the only way of implementing these services. For all services there are off-line methods of achieving the same result, and this Recommendation | International Standard does not mandate use of on-line protocols. For example, when hardware tokens are used, many of the services may be achieved as part of the physical token delivery.

NOTE 2 – The ASN.1 code is equivalent to that in the RFCs mentioned above, though the syntax looks different in parts.

Basically, each TTP information exchange type consists of a request-message sent forward by the initiator and a response-message sent back by the responder, as shown in Figure 2. In case of problems, the response may be substituted by an error-message.

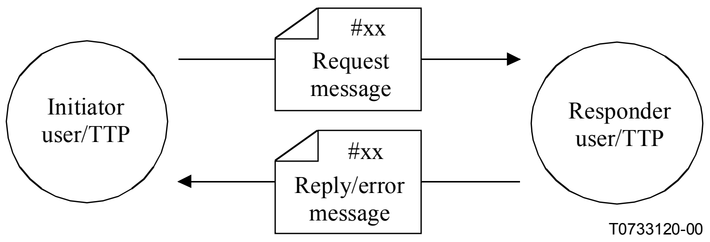

Figure 2 – Information exchange

Figure 3 shows the basic information flow in a TTP environment and all involved entities.

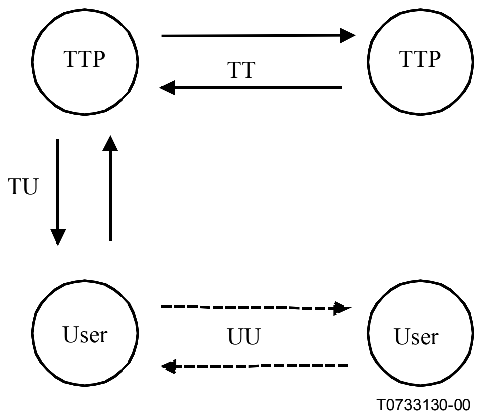

Figure 3 – Information flow

The following classes of TTP Information Exchange Types can be distinguished:

– TTP initiated information exchange types to TTPs: "TT"-class.

– TTP initiated information exchange types to end entities and vice versa: "TU"-class.

– End entity initiated information exchange types to other end entities: "UU"-class.

The "UU"-class of information exchange types is outside the scope of this Recommendation | International Standard.

7.1 Overview of certificate management services and messages

Note that not all PKI management services result in the creation of a PKI message.

The services including entities and CAs described below may additionally include a RA.

The message names printed in **bold** refer to the definitions in 8.1.2.

7.1.1 Initialization

7.1.1.1 CA initialization

Before issuing any certificates, a newly established CA (which plans to issue CRLs) shall produce initial revocation lists, i.e. "empty" versions of each CRL which is to be periodically produced.

If a newly created CA will serve as a directly trusted CA for some end entities, it shall produce at least one "self-certificate" which is a certificate for the CAs public key signed with the CA's own key. Different self-signed certificates, including different naming constraints, may be needed in order to support different applications. In order to make the CA's self-certificate useful to end entities that do not acquire the self-certificate via "out-of-band" means, the CA should also produce a hash-code for every self-signed certificate. End entities that acquire this hash-code and an identifier of the corresponding self-certificate securely via some "out-of-band" means can then verify this self-certificate and hence the other attributes contained therein.

From the perspective of PKI management protocols, the initialization of a CA, which is not a directly trusted CA, is the same as the initialization of an end entity. The only difference is that the CA must also produce an initial revocation list.

7.1.1.2 CA to CA initialization

Before entities of one CA can use the public keys of an entity certified by another CA, trust between these CAs must be established. This may be achieved by using a cross-certificate between these CAs or by means of a trusted certification path. This function may be supported by exchange mechanisms different from that used for entity interactions with the CA (e.g. physical exchange between CA managers), and so need not necessarily be supported by on-line exchange protocols.

Cross certification involves the use of three messages:

– Cross-certification request message (**CrossCertReq**);

– Cross-certification response message (**CrossCertRep**);

– PKI Confirm message (**PKIConfirm**).

The requester CA is the CA that will become the subject of the cross-certificate; the responder CA will become the issuer of the cross-certificate.

7.1.1.3 Entity initialization

This is the process whereby an end entity first makes itself known to a CA or RA, prior to the CA issuing a certificate, or certificates, for that end entity. The end result of this process (when it is successful) is that a CA issues a certificate for an end entity's public key, and returns that certificate to the end entity and/or posts that certificate in a public repository. This process may, and typically will, involve multiple "steps", possibly including an initialization of the end entity's equipment. For example, the end entity's equipment must be securely initialized with the public key of a CA, to be used in validating certificate paths. Furthermore, an end entity typically needs to be initialized with its own key pair(s).

The identity of the entity shall be verified by the RA. This registration procedure may involve off-line and even non-electronic communications (e.g. using the postal service) and need not be supported by any on-line exchange protocols. Various authentication schemes for initial registration of an end entity are described in 7.2.1.

Before using services offered by a TTP, an entity may need to obtain information about the functions supported by the TTP along with keys needed to communicate securely with the TTP and the TTP's public key needed to verify certificates.

The PKI GenMsg dialogue may be used to request and supply this PKI information. In this case, the request shall be the **GenMsg** message, the response shall be the **GenRep** message and the error response shall be the **ErrorMsg** message. These messages are protected using a MAC based on shared secret information (i.e. **PasswordBasedMAC**) or any other authenticated means (if the end entity has an existing certificate).

– PKI Information Request (**GenMsg** message);
– PKI Information Response (**GenRep** message).

This message pair is designed for general requests for PKI information and may be used after initialization too. It may also be used by other TTPs to acquire information about the current status of a CA. The CA shall respond to the request by providing the requested information, or by conveying an error if some of the information cannot be provided.

When carrying out the registration/certification process an entity may indicate that the request is its first certification request by using the following message instead of the **CertReq** and **CertRep** messages, as described later on:

– Initial Registration/Certification request (**InitReq**);
– Initial Registration/Certification response (**InitRep**).

7.1.2 Key generation

The public/private key pair used in an asymmetric system may be generated by either the entity, or a TTP. Wherever the key pair is generated, this must be done in a manner which can be trusted to properly generate it according to the algorithm used and maintain the integrity of both keys and the confidentiality of the private key.

a) Entity generated

If the entity generates the public/private key pair, the public key is passed to the TTP as part of the registration/certification process. This is included in a field of the certificate template contained in the **CertReqMessage**.

b) TTP generation

If the TTP generates the public/private key pair, this pair is passed to the entity as part of the registration/certification process (see 7.1.1.3). The private key is passed by physical PSE delivery or within the private key field of **CertKeyPair** within **CertRep**. The public key is passed within its certificate; the certificate field of **CertKeyPair** within **CertRep** is contained in **CertRepContent**.

c) Key update

When a key pair is due to expire, the relevant end entity may request a key update, that is, it may request that the TTP issues a new certificate for a new key pair. The request is made using a key update request message. If the end entity already possesses a signing key pair (with a corresponding verification certificate), then this message will typically be protected by the entity's digital signature. The TTP returns the new certificate (if the request is successful) in a key update response message:

– Certification request for an updated key (**KeyUpdReq**);
– Certification response for an updated key (**KeyUpdRep**).

The new key pair may be generated by the entity or by the TTP. In the second case the key is generated and carried in these messages as part of the registration/certification process as described above.

NOTE – In the case where the entity wishes to extend the life of an existing certified key, then the certificate needs to be updated.

d) CA key update announcement

CA keys (as all other keys) have a finite lifetime and will have to be updated on a periodic basis.

Special certificates (see 7.2.3) are issued by the CA to aid existing end entities who hold the old self-signed CA certificate to make a secure transition to the new self-signed CA certificate, and to aid new end entities who will hold the new self-signed CA certificate to acquire the old one securely for verification of existing data.

When a CA updates its own public/private key pair, the following messages may be used to announce this event to entities:

– Announce CA key update (**CAKeyUpdAnn**)

7.1.3 Key certification

An initialized end entity may request a certificate at any time (as part of an update procedure, or for any other purpose). This request will be made using the certification request message. If the end entity already possesses a signing key pair (with a corresponding verification certificate), then this message will typically be protected by the entity's digital signature. The CA returns the new certificate (if the request is successful) in a **CertRep** message.

An entity registers with a TTP and requests a certificate using the following messages:

- Registration/Certification request (**CertReq**);
- Registration/Certification response (**CertRep**).

An initial registration certification request/response (**InitReq/InitRep**) may optionally be used.

When the entity identifies the need for a new key pair and associated certificate, certification/registration requests may optionally be carried using the following messages:

- Registration/Certification request in the case of key update (**KeyUpdReq**);
- Registration/Certification response in the case of key update (**KeyUpdRep**).

The entity includes in the request any information which it requires to be placed in the certificate, including its name or a pseudonym. Certain information in the response, such as expiry time, may differ from that requested by the entity.

In order to complete the registration/certification process, further exchanges may be necessary to attest the validity of the entity's name and other information to be included in, or implied by, the certificate.

If the key pair is generated by the entity, then the public key may be passed to the TTP in the request. If the key pair is generated by the TTP, then the encrypted private key may be passed to the entity in the response.

If the key pair is generated by the entity, and if it is for generation and verification of digital signatures, then a signature may be included in the request to prove ownership of the private key.

If required, the entity may send the following message to demonstrate acceptance of the certificate.

- Confirm (**PKIConfirm**).

7.1.4 Certificate announcement

Having completed the registration/certification process including, if necessary, receipt of a **PKIConfirm** message, the TTP makes the certificate available to other parties. This may be achieved by a range of mechanisms including:

a) Placing the certificate in a repository such as a directory service or Web server;

b) Passing the certificate to other entities who are known to require the certificate (e.g. members of a known user community).

Alternatively, the entity may make its certificate available to other parties by including the certificate with any protected data.

A TTP may send certificates to entities using the following message:

- Certificate announce **CertAnn.**

The exchanges required to place certificates in a repository will depend on the form of repository used.

If a party requiring a certificate has not already been provided with it by a certificate announce message, or along with the protected data, then the certificate may be obtained from a repository such as a directory service, or Web server.

The exchanges required to obtain a certificate from a repository will depend on the form of repository used.

7.1.5 Key distribution

For key distribution, the private key needs to be treated differently from the public key.

Private key: For the private key, a differentiation needs to be made as to who generates the key pair. If an entity generates its own key pair, no distribution of the private key is necessary. If the key pair is not generated by the entity (i.e. it is generated by the TTP), the private key needs to be distributed to the entity in a secure way.

NOTE – A method for distributing private keys can be found in 8.4.4. For general information about the distribution of private keys, see ISO/IEC 11770 (all parts), which introduce different methods for key distribution.

Public key: The public keys of entities are distributed together with the certificates assigned to them. Associated messages are those concerning the announcement and retrieval of certificates.

7.1.6 Key/certificate revocation

Certificate revocation should take place whenever there is a suspicion that something has gone wrong with the keys (private key is compromised), or if changes relevant to the certificate associated with the public key occur (change of name, termination of employment in an organization, etc.). If an end entity (or any other authorized entity) wants to revoke the public key and the certificate associated with it, a request **RevReq** is sent to the TTP, which answers with a **RevRep**.

\If a TTP has revoked, or is about to revoke, the certificate requested, it may issue an announcement of this event (**RevAnn**). In particular, this might be the case if the request for revocation was not issued by the entity.

In combination with the revocation activities, the entries in the CRL change. To give notice of these changes, the TTP issues an announcement of the new CRL (**CRLAnn**). This information should be sent to all entities concerned.

7.2 Assumptions and restrictions for some of the services

7.2.1 Initial registration/certification

There are many schemes that can be used to achieve initial registration and certification of end entities. No one method is suitable for all situations due to the range of policies which a CA may implement, and the variation in the types of end entity which can occur.

The initial registration/certification schemes that are supported by this Recommendation | International Standard can, however, be classified. This classification is done for the situation where the end entity in question has had no previous contact with the PKI. Where the end entity already possesses certified keys, then some simplifications/alternatives are possible.

The following criteria may be used to distinguish various schemes:

a) Authentication to the RA

During initial registration/certification, the end entity deals with the RA.

It may be independent or combined with the CA. The RA may be needed to authenticate the requester at some time before issuing a certificate. This authentication may happen either before any message is exchanged with the RA, or after some messages have been exchanged with the RA. This may be done through a direct contact (e.g. the presentation of an ID card, a driving licence, a passport) or using other out-of-band means (e.g. call-back procedure, voice recognition).

b) Initial information distributed by out-of-band means

Public and/or secret information needs to be distributed by out-of-band means before any protocol may be used. Public information may consist of the name of an RA (or of a CA); its location and its public key value and algorithm or a pointer to a self-signed certificate and a hash of it. Secret information may consist of an identifier for the end entity and an initial authentication key. The initial authentication key can then be used to protect relevant PKI messages.

The public information may be freely shared by all users of a community and assurance of its integrity is sufficient. As an example, that information may be on a CD-ROM.

The end entity, or the organization the end entity represents, will be given the secret information as a result of the initial authentication (see above). The secret part of that information will need to be protected against disclosure.

c) End entity message origin authentication

The on-line messages produced by the end entity that requires a certificate and sent by the end entity to the RA or CA, may be authenticated or not. An initial registration/certification procedure can be secure when the messages from the end entity are authenticated via some out-of-band means (e.g. a subsequent visit).

d) Location of key generation

There are three possibilities for the location of key pair generation: the end entity, an RA, or a CA.

NOTE – This does not preclude an independent key generation service. The actual key pair may have been generated elsewhere and transported to the end entity, RA, or CA using a (proprietary or standardized) key generation request/response protocol (outside the scope of this Recommendation | International Standard).

e) Confirmation of successful certification

Following the creation of an initial certificate for an end entity, additional assurance can be gained by having the end entity explicitly confirm successful receipt of the message containing (or indicating the creation of) the certificate. Naturally, this confirmation message must be protected (based on an initial authentication key or other means). This gives two further possibilities: confirmed or not.

The criteria above allow for a large number of initial registration/certification schemes. A few of them are described here.

7.2.1.1 Centralized scheme

In terms of the criteria above, this scheme is where:

- initial authentication occurs at the RA before any exchange;
- secret information is distributed by out-of-band means by the RA during the initial authentication;
- no on-line message from the end entity is required;
- key pair generation occurs at the certifying CA or RA;
- no confirmation message is required.

In terms of message flow, since key pair generation occurs at the certifying CA or RA, the PSE must be returned to the end entity. The only message required is sent from the CA to the end entity and contains the entire PSE for the end entity. The secret information distributed previously by out-of-band means allows the end entity to authenticate the message received and decrypt any encrypted values. An alternative is the physical delivery of a PSE token.

7.2.1.2 Pre-authenticated scheme

In terms of the criteria above, this scheme is where:

- initial authentication occurs at the RA before any exchange;
- secret information is distributed by out-of-band means by the RA during the initial authentication;
- messages from the end entity are authenticated using this secret information;
- key pair generation occurs at the end entity;
- a confirmation message is required.

In terms of message flow, this scheme is as follows:

End entity		**RA/CA**
	Authentication of the end entity to the RA.	
	Out-of-band distribution of Initial Authentication Key (IAK) and reference value (RA/CA → EE) by the RA.	
Key generation Creation of certification request Protect request with IAK		
	→→certification request→→	
		Verify request Process request Create certificate Create response
	←←certification response←←	
Handle response Create confirmation		
	→→confirmation message→→	Verify confirmation

(Where verification of the confirmation message fails, the RA/CA shall revoke the newly issued certificate if necessary.)

7.2.1.3 Post-authenticated scheme

In terms of the criteria above, this scheme is where:

- initial authentication occurs at the RA after the exchanges;
- public information about the RA is distributed by out-of-band means (i.e. no secret information is distributed);
- messages from the end entity are not authenticated;
- key pair generation occurs at the end entity;
- a confirmation message is not required.

In terms of message flow, this scheme is as follows:

End entity		**RA/CA**
	Out-of-band distribution of public information (RA/CA → EE)	
Key generation Creation of certification request No protection of the request		
	→>>–certification request→>>–	
		Verify request Process request Create response
	–<<–certification response–<<–	
Handle response Verify the origin of the response Extract the registration number		
	Presentation of the registration number and authentication of the end entity to the RA using out-of-band means.	
		Create certificate

The end entity may at any time fill in a request as soon as it possesses the right information about the RA (in particular, its public key). The RA allocates a unique registration number for any unauthenticated certification request received. Since the certification response issued by the RA is signed, the end entity may verify its origin and extract the registration number. The end entity may then present that number to the RA and authenticate itself using some out-of-band means.

The benefits of this scheme are that:

– no contact with the RA is required in advance;

– no secret information is ever manipulated.

7.2.2 Proof of Possession (POP) of Private Key

In order to prevent certain attacks and to allow a CA/RA to properly check the validity of the binding between an end entity and a signature key pair, the PKI management operations specified here make it possible for an end entity to prove that it has possession of (i.e. is able to use) the private signature key corresponding to the public signature key for which a certificate is requested. A given CA/RA is free to choose how to enforce POP (e.g. out-of-band procedural means versus certificate management protocol ("in-band") messages) in its certification exchanges (i.e. this may be a policy issue). However, conforming CAs/RAs shall force POP by some means.

This Recommendation | International Standard explicitly allows for cases where an end entity supplies the relevant proof to an RA and the RA subsequently attests to the CA that the required proof has been received and validated. For example, an end entity wishing to get a certificate for a signing key could send the appropriate signature to the RA which then simply notifies the relevant CA that the end entity has supplied the required proof. Of course, such a situation may be disallowed by some policies (e.g. CAs may be the only entities permitted to verify POP during certification).

Since the key pair of the entity is generated for digital signature purposes, the end entity can sign a suitable value to prove possession of the private key. The value has to be chosen in a way preventing attackers from successfully using old signatures of the end entity.

7.2.3 Directly trusted CA key update

The basis of the procedure described here is that the CA protects its new public key using its previous private key and vice versa. Thus when a CA updates its key pair it shall generate two extra **cACertificate** attribute values if certificates are made available using an X.500 directory (for a total of four: **OldWithOld**; **OldWithNew**; **NewWithOld**; and **NewWithNew**).

When a CA changes its key pair, those entities who have acquired the old CA public key via "out-of-band" means are most affected. It is these end entities which will need access to the new CA public key protected with the old CA private key. However, they will only require this for a limited period (until they have acquired the new CA public key via the "out-of-band" mechanism). This will typically be easily achieved when these end entities' certificates expire.

The data structure used to protect the new and old CA public keys is a standard certificate (which may also contain extensions). There are no new data structures required.

NOTE 1 – This scheme does not make use of any of the X.509 v3 or v4 extensions as it is designed to work even for version 1 certificates. The presence of the **KeyIdentifier** extension could be used for efficiency improvements.

NOTE 2 – While the scheme could be generalized to cover cases where the CA updates its key pair more than once during the validity period of one of its end entities' certificates, this generalization seems of dubious value. Not having this generalization simply means that the validity period of a CA key pair must be greater than the validity period of any certificate issued by that CA using that key pair.

NOTE 3 – This scheme forces end entities to acquire the new CA public key on the expiry of the last certificate they owned that was signed with the old CA private key (via the "out-of-band" means). Certificate and/or key update services occurring at other times do not necessarily require this (depending on the end entity's equipment).

CA – actions

To change the key of the CA, the CA does the following:

1) Generate a new key pair;
2) Create a certificate containing the old CA public key signed with the new private key (the "old with new" certificate);

   This allows end entities whose PSE contain the new public key of the CA to verify old CA key and certificates signed with this key.
3) Create a certificate containing the new CA public key signed with the old private key (the "new with old" certificate);

   This allows end entities whose PSE contain the old public key of the CA to verify the new CA key and certificates signed with this key.
4) Create a certificate containing the new CA public key signed with the new private key (the "new with new" certificate);

   This allows end entities whose PSE contain the old public key of the CA to import this new self signed certificate.
5) Publish these new certificates via the directory and/or other means (perhaps using a CAKeyUpdAnn message);
6) Export the new CA public key so that end entities may acquire it using the "out-of-band" mechanism (if required).

The old CA private key is then no longer required. The old CA public key will, however, remain in use for some time. The time when the old CA public key will no longer be required (other than for non-repudiation) will be when all end entities of this CA have securely acquired the new CA public key.

The "old with new" certificate shall have a validity period starting at the generation time of the old key pair and ending at the expiry date of the old public key.

The "new with old" certificate shall have a validity period starting at the generation time of the new key pair and ending at the time by which all end entities of this CA will securely possess the new CA public key (at the latest, the expiry date of the old public key).

The "new with new" certificate shall have a validity period starting at the generation time of the new key pair and ending at the time by which the CA will next update its key pair.

The procedure of publishing these certificates is useful for revocation checks too as the CA may have signed the CRL using a newer private key than the one that is within the entity's PSE.

7.2.4 Cross-certification

The following subclause describes a possible scheme for cross-certification. However, other schemes may be used depending on the policies of the TTPs encountered in cross-certification. For a discussion of possible scenarios, see Annex A.

7.2.4.1 One-way request-response scheme

The cross-certification scheme is essentially a one-way service; that is, when successful, this service results in the creation of one new cross-certificate. If the requirement is that cross-certificates be created in "both directions", then each CA in turn must initiate a cross-certification service (or use another scheme).

This scheme is suitable where the two CAs in question can already verify each other's signatures (they have some common points of trust), or where there is an out-of-band verification of the origin of the certification request.

Detailed Description

Cross-certification is initiated by CA1. The CA1 identifies the CA (i.e. CA2) it wants to cross-certify and the CA1 equipment generates an authorization code. The CA1 passes this authorization code by out-of-band means to the CA2. The CA2 enters the authorization code in order to initiate the on-line exchange.

The authorization code is used for authentication and integrity purposes. This is done by generating a symmetric key based on the authorization code and using the symmetric key for generating Message Authentication Codes (MACs) on all messages exchanged.

CA2 initiates the exchange by generating a random number (requester random number). CA2 then sends to CA1 the cross-certification request message (**CrossCertReq**). The fields in this message are protected from modification with a MAC based on the authorization code.

Upon receipt of the cross-certification request message, CA1 checks the protocol version, saves the requester random number, generates its own random number (responder random number) and validates the MAC. It then generates (and archives, if desired) a new requester certificate that contains the CA2 public key and is signed with the CA1 signature private key. CA1 responds with the cross-certification response message (**CrossCertRes**). The fields in this message are protected from modification with a MAC based on the authorization code.

Upon receipt of the cross-certification response message, CA2 checks that its own system time is close to the CA1 system time, checks the received random numbers and validates the MAC. CA2 responds with the **PKIConfirm** message. The fields in this message are protected from modification with a MAC based on the authorization code. CA2 writes the requester certificate to the Repository.

Upon receipt of the **PKIConfirm** message, CA1 checks the random numbers and validates the MAC.

NOTE 1 – The cross-certification request message must contain a "complete" certification request, that is, all fields (including, for example, a **BasicConstraints** extension) must be specified by CA2.

NOTE 2 – The cross-certification response message should contain the verification certificate of CA1, if present, CA2 must then verify this certificate (for example, via the "out-of-band" mechanism).

8 Data structures for certificate management messages

This clause contains descriptions of the data structures required for certificate management messages. The data structures will be specified in accordance with RFC documents RFC 2510 and RFC 2511. This Recommendation | International Standard uses ASN.1 syntax and imports the ASN.1definitions from ITU-T Rec. X.509 | ISO/IEC 9594-8.

8.1 Overall message

All the messages used in this Recommendation | International Standard for the purposes of certificate management use the following structure:

```
PKIMessage ::= SEQUENCE {
    header        PKIHeader,
    body          PKIBody,
    protection    [0] PKIProtection    OPTIONAL,
    extraCerts    [1] SEQUENCE SIZE (1..MAX) OF Certificate OPTIONAL
}
```

The PKI message header, body and protection will be discussed in detail in the subclauses below.

The extra certificates field **extraCerts** can contain certificates which may be useful to the recipient. For example, this can be used by a TTP to present an end entity with certificates which it needs to verify its own new certificate (if the TTP that issued the end entity's certificate is not a directly trusted CA for the end entity). It should also be noted that this field does not necessarily contain a certification path. The recipient may have to sort, select from, or otherwise process the extra certificates in order to use them.

8.1.1 PKI Message Header

All TTP messages require some header information for addressing and transaction identification. Some of this information may also be present in a transport-specific envelope; however, if the message is protected then this information is also protected (i.e. no assumption about secure transport is made).

The following data structure is used to contain this information:

```
PKIHeader ::= SEQUENCE {
                    pvno           INTEGER    { version2 (1) },
                    sender            GeneralName,
                    recipient      GeneralName,
                    messageTime    [0] GeneralizedTime       OPTIONAL,
                    protectionAlg  [1] AlgorithmIdentifier   OPTIONAL,
                    senderKID      [2] KeyIdentifier         OPTIONAL,
                    recipKID       [3] KeyIdentifier         OPTIONAL,
                    transactionID  [4] OCTET STRING          OPTIONAL,
                    senderNonce    [5] OCTET STRING          OPTIONAL,
                    recipNonce     [6] OCTET STRING          OPTIONAL,
                    freeText       [7] PKIFreeText           OPTIONAL,
                    generalInfo    [8] SEQUENCE SIZE (1..MAX) OF InfoTypeAndValue OPTIONAL
                              -- this may be used to convey context-specific information
}
```

The **pvno** field is fixed for this version of the TTP document. The **sender** field contains the name of the sender of the **PKIMessage**. This name (in conjunction with **senderKID**, if supplied) should be usable to verify the protection on the message.

If nothing about the sender is known to the sending entity (e.g. DN, e-mail name, IP address, etc.), then the "sender" field shall contain a "NULL" value; that is, the SEQUENCE OF relative distinguished names is of zero length. In such a case, the **senderKID** field shall hold an identifier (i.e. a reference number) which indicates to the receiver the appropriate shared secret information to use to verify the message.

The **recipient** field contains the name of the recipient of the **PKIMessage**. This name (in conjunction with **recipKID**, if supplied) should be used to verify the protection granted to the message. The **messageTime** field contains the time at which the sender created the message (used when sender believes that the transport will be "suitable"; i.e. that the time will still be meaningful upon receipt). This may be useful to allow end entities to correct their local time to be consistent with the time on a central system. The **protectionAlg** field specifies the algorithm used to protect the message. If no protection bits are supplied (**PKIProtection** is optional), then this field shall be omitted; if protection bits are supplied, then this field shall be supplied.

The fields **senderKID** and **recipKID** are used to indicate which keys have been used to protect the message. The **transactionID** field within the message header is useful for the recipient of a response message who can correlate this with a previously issued request. For example, in the case of an RA there may be many requests outstanding at a given moment.

The **senderNonce** and **recipNonce** fields protect the **PKIMessage** against replay attacks. Nonces are used to provide replay protection, senderNonce is inserted by the creator of this message; **recipNonce** is a nonce previously inserted in a related message by the intended recipient of this message. The **freeText** field may be used to send a human-readable message to the recipient. The structure of this field is:

```
PKIFreeText ::= SEQUENCE SIZE (1..MAX) OF UTF8String
            -- text encoded as UTF-8 String (NOTE – Each UTF8String should
            -- include an RFC 1766 language tag to indicate the language
            -- of the contained text)
```

8.1.2 PKI Message Body

```
PKIBody ::= CHOICE {   -- message-specific body elements
    ir        [0]  CertReqMessages,       --Initialization Request (InitReq)
    ip        [1]  CertRepMessage,        --Initialization Response (InitRep)
    cr        [2]  CertReqMessages,       --Certification Request (CertReq)
    cp        [3]  CertRepMessage,        --Certification Response (CertRep)
    p10cr     [4]  CertificationRequest,  --PKCS #10 Cert.- Request (alternative form for CertRep)
                   -- the certification request as defined in PKCS #10 for compatibility
    kur       [7]  CertReqMessages,        --Key Update Request (KeyUpdReq)
    kup       [8]  CertRepMessage,         --Key Update Response (KeyUpdRep)
    rr        [11] RevReqContent,          --Revocation Request (RevReq)
    rp        [12] RevRepContent,          --Revocation Response (RevRep)
    ccr       [13] CertReqMessages,        --Cross-Cert. Request (CrossCertReq)
    ccp       [14] CertRepMessage,         --Cross-Cert. Response (CrossCertRep)
    ckuann    [15] CAKeyUpdAnnContent,     --CA Key Update Ann. (CAKeyUpdAnn)
    cann      [16] CertAnnContent,         --Certificate Ann. (CertAnn)
    rann      [17] RevAnnContent,          --Revocation Ann. (RevAnn)
    crlann    [18] CRLAnnContent,          --CRL Announcement (CRLAnn)
    conf      [19] PKIConfirmContent,      --PKI Confirmation (PKIConfirm)
    nested    [20] NestedMessageContent,   --Nested Message
    genm      [21] GenMsgContent,          --General Message (GenMsg)
    genp      [22] GenRepContent,          --General Response (GenRep)
    error     [23] ErrorMsgContent,        --Error Message (ErrorMsg)
}
```

The specific types are described in 8.3 and 8.4 below.

8.1.3 PKI Message Protection

Some PKI messages will be protected for integrity. (Note that if an asymmetric algorithm is used to protect a message and the relevant public component has been certified already, then the origin of message can also be authenticated. On the other hand, if the public component is uncertified, then the message origin cannot be automatically authenticated, but may be authenticated via out-of-band means.)

When protection is applied the following structure is used:

```
PKIProtection ::= BIT STRING
```

The input to the calculation of PKIProtection is the DER encoding of the following data structure:

```
ProtectedPart ::= SEQUENCE {
    header    PKIHeader,
    body      PKIBody
}
```

There may be cases in which the PKIProtection BIT STRING is deliberately not used to protect a message (i.e. this OPTIONAL field is omitted) because other protection, external to this Recommendation | International Standard, will be applied instead. Such a choice is explicitly allowed in this Recommendation | International Standard.

It is noted, however, that many such external mechanisms require that the end entity already possesses a public-key certificate, and/or a unique Distinguished Name, and/or other such infrastructure-related information. Thus, they may not be appropriate for initial registration or any other process with "boot-strapping" characteristics. For those cases, it may be necessary that the PKIProtection parameter be used.

Depending on the circumstances the PKIProtection bits may contain a Message Authentication Code (MAC) or signature. The following cases may occur:

Case 1: shared secret information

In this case, the sender and recipient share secret information (established via out-of-band means or from a previous TTP management service). PKIProtection will contain a MAC value.

The protectionAlg will be the following:

```
PasswordBasedMac ::= OBJECT IDENTIFIER     --{1 2 840 113533 7 66 13}
PBMParameter ::= SEQUENCE {
    salt              OCTET STRING,
    owf               AlgorithmIdentifier,
    -- AlgorithmIdentifier for a One-Way Function (SHA-1 recommended)
    iterationCount    INTEGER,
    -- number of times the OWF is applied
    mac               AlgorithmIdentifier
    -- the MAC AlgorithmIdentifier (e.g. DES-MAC, Triple-DES-MAC as in PKCS #11,
}   -- or HMAC as in RFC2104, RFC2202)
```

In the above **protectionAlg** the salt value is appended to the shared secret input. The OWF is then applied **iterationCount** times, where the salted secret is the input to the first iteration and, for each successive iteration, the input is set to be the output of the previous iteration. The output of the final iteration (called "BASEKEY" for ease of reference, with a size of "H") is what is used to form the symmetric key. If the MAC algorithm requires a K-bit key and K <= H, then the most significant K bits of BASEKEY are used. If K > H, then all of BASEKEY is used for the most significant H bits of the key, OWF("1" || BASEKEY) is used for the next most significant H bits of the key, OWF("2" || BASEKEY) is used for the next most significant H bits of the key, and so on, until all K bits have been derived. [Here "N" is the ASCII byte encoding the number N and "||" represents concatenation.]

Case 2: Diffie-Hellman key pairs

Where the sender and receiver possess Diffie-Hellman certificates with compatible DH parameters, then in order to protect the message the end entity must generate a symmetric key based on its private DH key value and the DH public key of the recipient of the message. PKIProtection will contain a MAC value keyed with this derived symmetric key.

```
DHBasedMac ::= OBJECT IDENTIFIER --{1 2 840 113533 7 66 30}
  DHBMParameter ::= SEQUENCE {
    owf         AlgorithmIdentifier,
-- AlgorithmIdentifier for a One-Way Function (SHA-1 recommended)
    mac         AlgorithmIdentifier
    -- the MAC AlgorithmIdentifier  (e.g. DES-MAC, Triple-DES-MAC as in PKCS#11,
}   -- or HMAC as in RFC2104, RFC2202)
```

In the above protectionAlg OWF is applied to the result of the Diffie-Hellman computation. The OWF output (called "BASEKEY" for ease of reference, with a size of "H") is what is used to form the symmetric key. If the MAC algorithm requires a K-bit key and K <= H, then the most significant K bits of BASEKEY are used. If K > H, then all of BASEKEY is used for the most significant H bits of the key, OWF("1" || BASEKEY) is used for the next most significant H bits of the key, OWF("2" || BASEKEY) is used for the next most significant H bits of the key, and so on, until all K bits have been derived. [Here "N" is the ASCII byte encoding the number N and "||" represents concatenation.]

Case 3: digital signature

Where the sender possesses a signature key pair it may simply sign the message. **PKIProtection** will contain the signature value and the **protectionAlg** will be an **AlgorithmIdentifier** for a digital signature (e.g. **md5WithRSAEncryption** or **dsaWithSha-1**).

Case 4: multiple protection

In cases where an end entity sends a protected message to an RA, the RA may forward that message to a CA, attaching its own protection (which may be a MAC or a signature, depending on the information and certificates shared between the RA and the CA). This is accomplished by nesting the entire message sent by the end entity within a new TTP message. The structure used is as follows.

```
NestedMessageContent ::= PKIMessage
```

8.2 Common Data Structures

In this subclause some data structures are defined that are used in more than one PKI message.

8.2.1.1 Requested Certificate Contents

Various TTP management messages require that the originator of the message indicate some of the fields that are required to be present in a certificate. The **CertTemplate** structure allows an end entity or RA to specify as much as it wishes about the certificate it requires. **CertTemplate** is identical to a Certificate but with all fields optional.

Note that even if the originator completely specifies the contents of a certificate it requires, a CA is free to modify fields within the certificate actually issued.

See 8.3 for **CertTemplate** syntax.

8.2.2 Encrypted Values

Where encrypted values (restricted, in this Recommendation | International Standard, to be either private keys or certificates) are sent in PKI messages, the **EncryptedValue** data structure is used.

See 8.6.1 for **EncryptedValue** syntax.

Use of this data structure requires that the creator, and intended recipient respectively, be able to encrypt and decrypt. Typically, this will mean that the sender and recipient have, or are able to generate, a shared secret key.

If the recipient of the PKIMessage already possesses a private key usable for decryption, then the **encSymmKey** field may contain a session key encrypted using the recipient's public key.

8.2.3 Status codes and failure Information for PKI messages

All response messages will include some status information. The following values are defined:

```
PKIStatus ::= INTEGER {
    granted                  (0),
    -- you got exactly what you asked for
    grantedWithMods          (1),
    -- you got something like what you asked for; the
    -- requester is responsible for ascertaining the differences
    rejection                (2),
    -- you don't get it, more information elsewhere in the message
    waiting                  (3),
    -- the request body part has not yet been processed,
    -- expect to hear more later
    revocationWarning        (4),
    -- this message contains a warning that a revocation is
    -- imminent
    revocationNotification   (5),
    -- notification that a revocation has occurred
    keyUpdateWarning         (6)
    -- update already done for the oldCertId specified in
    -- the key update request message
}
```

Responders may use the following syntax to provide more information about failure cases.

```
PKIFailureInfo ::= BIT STRING {
-- since a request can fail in more than one way!
-- More codes may be added in the future if/when required.
   badAlg                    (0),
   -- unrecognized or unsupported Algorithm Identifier
   badMessageCheck           (1),
   -- integrity check failed (e.g. signature did not verify)
   badRequest                (2),
   -- transaction not permitted or supported
   badTime                   (3),
   -- messageTime was not sufficiently close to the system time,
   -- as defined by local policy
   badCertId                 (4),
   -- no certificate could be found matching the provided criteria
   badDataFormat             (5),
   -- the data submitted has the wrong format
   wrongAuthority            (6),
   -- the authority indicated in the request is different from the
   -- one creating the response token
   incorrectData             (7),
   -- the requester's data is incorrect (used for notary services)
   missingTimeStamp          (8)
   -- when the timestamp is missing but should be there (by policy)
}


PKIStatusInfo ::= SEQUENCE {
   status           PKIStatus,
   statusString     PKIFreeText     OPTIONAL,
   failInfo         PKIFailureInfo  OPTIONAL
}
```

8.2.4 Certificate Identification

In order to identify particular certificates the **CertId** data structure is used. See 8.3 for **CertId** syntax.

8.2.5 "Out-of-band" directly trusted CA public key

Each directly trusted CA must be able to publish its current public key via some "out-of-band" means. While such mechanisms are beyond the scope of this Recommendation | International Standard, data structures which can support such mechanisms are defined.

There are generally two methods available: either the CA directly publishes its self-signed certificate; or this information is available via the Directory (or equivalent) and the CA publishes a hash-code of this value to allow verification of its integrity before use.

OOBCert ::= Certificate

The fields within this certificate are restricted as follows:

- The certificate shall be self-signed (i.e. the signature must be verifiable using the SubjectPublicKeyInfo field);
- The subject and issuer fields shall be identical;
- If the subject field is **NULL**, then both **subjectAltNames** and **issuerAltNames** extensions shall be present and have exactly the same value;
- The values of all other extensions shall be suitable for a self-signed certificate (e.g. key identifiers for subject and issuer shall be the same).

```
OOBCertHash ::= SEQUENCE {
   hashAlg      [0] AlgorithmIdentifier      OPTIONAL,
   certId       [1] CertId                   OPTIONAL,
   hashVal      BIT STRING
   -- hashVal is calculated over the self-signed
   -- certificate with the identifier certID.
}
```

The intention of the hash-code is that anyone who has securely received the hash-code (via the out-of-band means) can verify a self-signed certificate for that CA.

8.2.6 Publication Information

Requesters may indicate that they wish the PKI to publish a certificate using the **PKIPublicationInfo** structure. See 8.3 for **PKIPublicationInfo** syntax.

8.2.7 Proof-of-Possession Structures

The proof of possession of the private signing key is demonstrated through use of the **POPOSigningKey** structure. See 8.3 for **POPOSigningKey** syntax, but note that POPOSigningKeyInput has the following semantic stipulations in this specification:

```
POPOSigningKeyInput ::= SEQUENCE {
        authInfo        CHOICE {
            sender                [0] GeneralName,
            -- from PKIHeader (used only if an authenticated identity
            -- has been established for the sender (e.g. a DN from a
            -- previously-issued and currently-valid certificate))
            publicKeyMAC     [1] PKMACValue
            -- used if no authenticated GeneralName currently exists for
            -- the sender; publicKeyMAC contains a password-based MAC
            -- (using the protectionAlg AlgorithmIdentifier from PKIHeader) on the
            -- DER-encoded value of publicKey
            },
        publicKey           SubjectPublicKeyInfo-- from CertTemplate
        }
```

8.3 Data structures specific for Certificate Request Messages of type CertReq

This subclause describes the Certificate Request Message Format (CRMF). This syntax is used to convey a request for a certificate to a Certification Authority (CA) (possibly via a Registration Authority (RA)) for the purposes of X.509 certificate production. The request will typically include a public key and associated registration information.

8.3.1 Overview

Construction of a certification request involves the following steps:

a) A CertRequest value is constructed. This value may include the public key, all or a portion of the end-entity's (EEs) name, other requested certificate fields, and additional control information related to the registration process.

b) A proof of possession (of the private key corresponding to the public key for which a certificate is being requested) value may be calculated across the CertRequest value.

c) Additional registration information may be combined with the proof of possession value and the CertRequest structure to form a CertReqMessage.

d) The CertReqMessage is securely communicated to a CA (as specified in 8.1.3).

8.3.2 CertReqMessage Syntax

A certificate request message body is composed of the certificate request, an optional proof of possession field and an optional registration information field.

```
CertReqMessages ::= SEQUENCE SIZE (1..MAX) OF CertReqMsg

CertReqMsg ::= SEQUENCE {
   certReq  CertRequest,
   pop      ProofOfPossession  OPTIONAL,
   regInfo  SEQUENCE SIZE(1..MAX) OF AttributeTypeAndValue OPTIONAL }
```

The proof of possession field is used to demonstrate that the entity to be associated with the certificate is actually in possession of the corresponding private key. This field may be calculated across the contents of the **certReq** field and varies in structure and content by public key algorithm type and service mode.

The **regInfo** field should only contain supplementary information related to the context of the certification request when such information is required to fulfill a certification request. This information may include subscriber contact information, billing information or other ancillary information useful to fulfillment of the certification request.

Information directly related to certificate content should be included in the **certReq** content. However, inclusion of additional **certReq** content by RAs may invalidate the pop field. Data therefore intended for certificate content may be provided in **regInfo**.

8.3.3 Proof of Possession (POP)

In order to prevent certain attacks and to allow a CA/RA to properly check the validity of the binding between an end entity and a key pair, the PKI management services specified here make it possible for an end entity to prove that it has possession of (i.e. is able to use) the private key corresponding to the public key for which a certificate is requested. A given CA/RA is free to choose how to enforce POP (e.g. out-of-band procedural means versus the CRMF in-band message) in its certification exchanges (i.e. this may be a policy issue). However, CAs/RAs shall enforce POP by some means.

This Recommendation | International Standard allows for cases where POP is validated by the CA, the RA, or both. Some policies may require the CA to verify POP during certification, in which case the RA shall forward the end entity's CertRequest and ProofOfPossession fields unaltered to the CA, and as an option may also verify POP. If the CA is not required by policy to verify POP, then the RA should forward the end entity's request and proof unaltered to the CA as above. If this is not possible (for example, because the RA verifies POP by an out-of-band method), then the RA may attest to the CA that the required proof has been validated. If the CA uses an out-of-band method to verify POP (such as physical delivery of CA-generated private keys), then the ProofOfPossession field is not used.

8.3.3.1 Proof of Possession Syntax

```
ProofOfPossession ::= CHOICE {
   raVerified      [0] NULL,
   -- used if the RA has already verified that the requester is in
   -- possession of the private key
   signature       [1] POPOSigningKey,
}

POPOSigningKey ::= SEQUENCE {
   poposkInput            [0] POPOSigningKeyInput OPTIONAL,
   algorithmIdentifier    AlgorithmIdentifier,
   signature              BIT STRING }
```

```
    -- The signature (using "algorithmIdentifier") is on the
    -- DER-encoded value of poposkInput. NOTE – If the CertReqMsg
    -- certReq CertTemplate contains the subject and publicKey values,
    -- then poposkInput shall be omitted and the signature shall be
    -- computed on the DER-encoded value of CertReqMsg certReq. If
    -- the CertReqMsg certReq CertTemplate does not contain the public
    -- key and subject values, then poposkInput shall be present and
    -- shall be signed. This strategy ensures that the public key is
    -- not present in both the poposkInput and CertReqMsg certReq CertTemplate fields.

POPOSigningKeyInput ::= SEQUENCE {
    authInfo        CHOICE {
        sender      [0] GeneralName,
        -- used only if an authenticated identity has been
        -- established for the sender (e.g. a DN from a
        -- previously-issued and currently-valid certificate)
        publicKeyMAC      PKMACValue },
        -- used if no authenticated GeneralName currently exists for
        -- the sender; publicKeyMAC contains a password-based MAC
        -- on the DER-encoded value of publicKey
    publicKey       SubjectPublicKeyInfo }  -- from CertTemplate

PKMACValue ::= SEQUENCE {
    algId  AlgorithmIdentifier,
    -- the algorithm value shall be PasswordBasedMac
    -- {1 2 840 113533 7 66 13}
    -- the parameter value is PBMParameter
    value  BIT STRING }
```

8.3.3.2 Use of Password-Based MAC

The following algorithm shall be used when **publicKeyMAC** is used in **POPOSigningKeyInput** to prove the authenticity of a request.

```
PBMParameter ::= SEQUENCE {
    salt            OCTET STRING,
    owf             AlgorithmIdentifier,
    -- AlgorithmIdentifier for a One-Way Function (SHA-1 recommended)
    iterationCount     INTEGER,
    -- number of times the OWF is applied
    mac             AlgorithmIdentifier
    -- the MAC AlgorithmIdentifier (e.g. DES-MAC, Triple-DES-MAC [PKCS#11],
}   -- or HMAC [RFC2104, RFC2202])
```

The process of using **PBMParameter** to compute **publicKeyMAC** and so authenticate the origin of a public key certification request consists of two stages. The first stage uses shared secret information to produce a MAC key. The second stage MACs the public key in question using this MAC key to produce an authenticated value.

Initialization of the first stage of algorithm assumes the existence of a shared secret distributed in a trusted fashion between CA/RA and end-entity. The salt value is appended to the shared secret and the one way function (owf) is applied iterationCount times, where the salted secret is the input to the first iteration and, for each successive iteration, the input is set to be the output of the previous iteration, yielding a key K.

In the second stage, K and the public key are inputs to HMAC as documented in RFC 2104 to produce a value for **publicKeyMAC** as follows:

publicKeyMAC = Hash(K XOR opad, Hash(K XOR ipad, public key))

where ipad and opad are defined in RFC 2104.

The AlgorithmIdentifier for owf shall be SHA-1 {1 3 14 3 2 26} and for mac shall be HMAC-SHA-1 {1 3 6 1 5 5 8 1 2}.

8.3.4 CertRequest syntax

The CertRequest syntax consists of a request identifier, a template of certificate content, and an optional sequence of control information.

```
CertRequest ::= SEQUENCE {
        certReqId     INTEGER,                 -- ID for matching request and response
        certTemplate  CertTemplate,            -- Selected fields of certificate to be issued
        controls      Controls OPTIONAL }      -- Attributes affecting issuance

CertTemplate ::= SEQUENCE {
    version          [0] Version               OPTIONAL,
    serialNumber     [1] INTEGER               OPTIONAL,
    signingAlg       [2] AlgorithmIdentifier   OPTIONAL,
    issuer           [3] Name                  OPTIONAL,
    validity         [4] OptionalValidity      OPTIONAL,
    subject          [5] Name                  OPTIONAL,
    publicKey        [6] SubjectPublicKeyInfo  OPTIONAL,
    issuerUID        [7] UniqueIdentifier      OPTIONAL,
    subjectUID       [8] UniqueIdentifier      OPTIONAL,
    xtensions        [9] Extensions            OPTIONAL }

OptionalValidity ::= SEQUENCE {
    notBefore   [0] Time OPTIONAL,
    notAfter    [1] Time OPTIONAL } --at least one shall be present

Time ::= CHOICE {
    utcTime      TCTime,
    generalTime  eneralizedTime }
```

8.3.5 Controls Syntax

The generator of a CertRequest may include one or more control values pertaining to the processing of the request.

```
Controls  ::= SEQUENCE SIZE(1..MAX) OF AttributeTypeAndValue
```

The following controls are defined (it is recognized that this list may expand over time): **regToken**; **authenticator**; **pkiPublicationInfo**; **oldCertID**; **protocolEncrKey**.

8.3.5.1 Registration Token Control

A **regToken** control contains one-time information (either based on a secret value or on knowledge) intended to be used by the CA to verify the identity of the subject prior to issuing a certificate. Upon receipt of a certification request containing a value for **regToken**, the receiving CA verifies the information in order to confirm the identity claimed in the certification request.

The value for **regToken** may be generated by the CA and provided out of band to the subscriber, or may otherwise be available to both the CA and the subscriber. The security of any out-of-band exchange should be commensurate with the risk of the CA accepting an intercepted value from someone other than the intended subscriber.

The **regToken** control would typically be used only for initialization of an end entity into the PKI, whereas the authenticator control (see next subclause) would typically be used for initial as well as subsequent certification requests.

In some instances of use the value for **regToken** could be a text string or a numeric quantity such as a random number. The value in the latter case could in principle be represented either as a binary quantity or as a text string. To ensure a uniform encoding of values regardless of the nature of the quantity, it is required that **regToken** shall be encoded as a value of type UTF8String.

8.3.5.2 Authenticator Control

An authenticator control contains information used on an ongoing basis to establish a non-cryptographic check of identity in communication with the CA. Examples include: mother's maiden name, last four digits of social security number, or other knowledge-based information shared with the subscriber's CA; a hash-code of such information; or other information produced for this purpose. The value for an authenticator control may be generated by the subscriber or by the CA.

In some instances of use the value for **authenticator** could be a text string or a numeric quantity such as a random number. The value in the latter case could in principle be represented either as a binary quantity or as a text string. To ensure a uniform encoding of values regardless of the nature of the quantity, it is required that **authenticator** shall be encoded as a value of type UTF8String.

8.3.5.3 Publication Information Control

The **pkiPublicationInfo** control enables subscribers to control the CA's publication of the certificate. It is defined by the following syntax:

```
PKIPublicationInfo ::= SEQUENCE {
    action      INTEGER {
                 dontPublish (0),
                 pleasePublish (1) },
    pubInfos  SEQUENCE SIZE (1..MAX) OF SinglePubInfo OPTIONAL }
     -- pubInfos SHALL not be present if action is "dontPublish"
     -- (if action is "pleasePublish" and pubInfos is omitted,
     -- "dontCare" is assumed)

SinglePubInfo ::= SEQUENCE {
    pubMethod    INTEGER {
        dontCare        (0),
        x500            (1),
        web             (2),
        ldap            (3) },
    pubLocation  GeneralName OPTIONAL }
```

If the **dontPublish** option is chosen, the requester indicates that the PKI should not publish the certificate (this may indicate that the requester intends to publish the certificate him/herself).

If the **dontCare** method is chosen, or if the **PKIPublicationInfo** control is omitted from the request, the requester indicates that the PKI may publish the certificate using whatever means it chooses.

If the requester wishes the certificate to appear in at least some locations but wishes to enable the CA to make the certificate available in other repositories, set two values of **SinglePubInfo** for **pubInfos**: one with **x500**, **web** or **ldap** value and one with **dontCare**.

The **pubLocation** field, if supplied, indicates where the requester would like the certificate to be found (note that the CHOICE within **GeneralName** includes a URL and an IP address, for example).

8.3.5.4 OldCert ID Control

If present, the **OldCertID** control specifies the certificate to be updated by the current certification request. The syntax of its value is:

```
CertId ::= SEQUENCE {
     issuer           GeneralName,
     serialNumber     INTEGER
}
```

8.3.5.5 Protocol Encryption Key Control

If present, the **protocolEncrKey** control specifies a key the CA is to use in encrypting a response to **CertReqMessages**. This control can be used when a CA has information to send to the subscriber that needs to be encrypted. Such information includes a private key generated by the CA for use by the subscriber.

The encoding of **protocolEncrKey** shall be **SubjectPublicKeyInfo**.

8.3.6 Object Identifiers

The OID **id-pkix** has the value

```
id-pkix  OBJECT IDENTIFIER  ::= { iso(1) identified-organization(3)
dod(6) internet(1) security(5) mechanisms(5) pkix(7) }

-- arc for Internet X.509 PKI protocols and their components
id-pkip  OBJECT IDENTIFIER :: { id-pkix pkip(5) }

-- Registration Controls in CRMF
id-regCtrl  OBJECT IDENTIFIER ::= { id-pkip regCtrl(1) }
id-regCtrl-regToken                OBJECT IDENTIFIER ::= { id-regCtrl 1 }
id-regCtrl-authenticator           OBJECT IDENTIFIER ::= { id-regCtrl 2 }
id-regCtrl-pkiPublicationInfo      OBJECT IDENTIFIER ::= { id-regCtrl 3 }
id-regCtrl-oldCertID               OBJECT IDENTIFIER ::= { id-regCtrl 5 }
id-regCtrl-protocolEncrKey         OBJECT IDENTIFIER ::= { id-regCtrl 6 }
```

```
-- Registration Info in CRMF
id-regInfo       OBJECT IDENTIFIER ::= { id-pkip id-regInfo(2) }
id-regInfo-asciiPairs    OBJECT IDENTIFIER ::= { id-regInfo 1 }
-- with syntax OCTET STRING
id-regInfo-certReq      OBJECT IDENTIFIER ::= { id-regInfo 2 }
-- with syntax CertRequest
```

8.4 Data structures specific for other messages

8.4.1 Initialization Request

An Initialization request message contains as the **PKIBody** an **CertReqMessages** data structure which specifies the requested certificate(s). Typically, **SubjectPublicKeyInfo**, **KeyId**, and **Validity** are the template fields which may be supplied for each certificate requested. This message is intended to be used for entities first initializing into the PKI.

See 8.3 for **CertReqMessages** syntax.

8.4.2 Initialization Response

An Initialization response message contains as the **PKIBody** an **CertRepMessage** data structure which has for each certificate requested a **PKIStatusInfo** field, a subject certificate, and possibly a private key (normally encrypted with a session key, which is itself encrypted with the **protocolEncKey**).

See 8.4.4 for **CertRepMessage** syntax.

8.4.3 Registration/Certification Request

A Registration/Certification request message contains as the **PKIBody** a **CertReqMessages** data structure which specifies the requested certificates. This message is intended to be used for existing PKI entities which wish to obtain additional certificates.

See 8.3 for **CertReqMessages** syntax.

Alternatively, the **PKIBody** may be a **CertificationRequest** (this structure is fully specified by the ASN.1 structure **CertificationRequest** given in PKCS#10). This structure may be required for certificate requests for signing key pairs when interservice with legacy systems is desired, but its use is strongly discouraged whenever not absolutely necessary.

8.4.4 Registration/Certification Response

A registration response message contains as the **PKIBody** a **CertRepMessage** data structure which has a status value for each certificate requested, and optionally has a CA public key, failure information, a subject certificate, and an encrypted private key.

```
CertRepMessage ::= SEQUENCE {
   caPubs         [1] SEQUENCE SIZE (1..MAX) OF Certificate OPTIONAL,
   response           SEQUENCE OF CertResponse
}

CertResponse ::= SEQUENCE {
   certReqId        INTEGER,
   -- to match this response with corresponding request (a value
   -- of –1 is to be used if certReqId is not specified in the
   -- corresponding request)
   status          PKIStatusInfo,
   certifiedKeyPair   CertifiedKeyPair   OPTIONAL,
   rspInfo            OCTET STRING     OPTIONAL
   -- analogous to the id-regInfo-asciiPairs OCTET STRING defined
   -- for regInfo in CertReqMsg A.1.3
}

CertifiedKeyPair ::= SEQUENCE {
   certOrEncCert      CertOrEncCert,
   privateKey       [0] EncryptedValue   OPTIONAL,
publicationInfo [1] PKIPublicationInfo     OPTIONAL
}
```

```
CertOrEncCert ::= CHOICE {
    certificate       [0] Certificate,
    encryptedCert      [1] EncryptedValue
}
```

Only one of the **failInfo** (in **PKIStatusInfo**) and **certificate** (in **CertifiedKeyPair**) fields can be present in each **CertResponse** (depending on the status). For some status values (e.g. waiting) neither of the optional fields will be present.

Given an **EncryptedCert** and the relevant decryption key, the certificate may be obtained. The purpose of this is to allow a CA to return the value of a certificate, but with the constraint that only the intended recipient can obtain the actual certificate. The benefit of this approach is that a CA may reply with a certificate even in the absence of a proof that the requester is the end entity which can use the relevant private key (note that the proof is not obtained until the **PKIConfirm** message is received by the CA). Thus the CA will not have to revoke that certificate in the event that something goes wrong with the proof of possession.

8.4.5 Key update request content

For key update requests the **CertReqMessages** syntax is used. Typically, **SubjectPublicKeyInfo**, **KeyId**, and **Validity** are the template fields which may be supplied for each key to be updated. This message is intended to be used to request updates to existing (non-revoked and non-expired) certificates.

See 8.3 for **CertReqMessages** syntax.

8.4.6 Key update response content

For key update responses the **CertRepMessage** syntax is used. The response is identical to the initialization response.

See 8.4.4 for **CertRepMessage** syntax.

8.4.7 Revocation request content

When requesting revocation of a certificate (or several certificates), the following data structure is used. The name of the requester is present in the **PKIHeader** structure.

```
RevReqContent ::= SEQUENCE OF RevDetails


RevDetails ::= SEQUENCE {
    certDetails       CertTemplate,
    -- allows requester to specify as much as they can about
    -- the cert. for which revocation is requested
    -- (e.g. for cases in which serialNumber is not available)
    revocationReason  ReasonFlags      OPTIONAL,
    -- the reason that revocation is requested
    badSinceDate      GeneralizedTime      OPTIONAL,
    -- indicates best knowledge of sender
    crlEntryDetails   Extensions       OPTIONAL
    -- requested crlEntryExtensions
}
```

8.4.8 Revocation response content

The response to the above message. If produced, this is sent to the requester of the revocation. (A separate revocation announcement message may be sent to the subject of the certificate for which revocation was requested.)

```
RevRepContent ::= SEQUENCE {
    status      SEQUENCE SIZE (1..MAX) OF PKIStatusInfo,
    -- in same order as was sent in RevReqContent
    revCerts  [0] SEQUENCE SIZE (1..MAX) OF CertId OPTIONAL,
    -- IDs for which revocation was requested (same order as status)
    crls    [1] SEQUENCE SIZE (1..MAX) OF CertificateList  OPTIONAL
    -- the resulting CRLs (there may be more than one)
}
```

8.4.9 Cross-certification request content

Cross-certification requests use the same syntax (**CertReqMessages**) as for normal certification requests with the restriction that the key pair shall be generated by the requesting CA and the private key shall not be sent to the responding CA.

See 8.3 for **CertReqMessages** syntax.

8.4.10 Cross-certification response content

Cross-certification responses use the same syntax (**CertRepMessage**) as for normal certification responses with the restriction that no encrypted private key can be sent.

See 8.4.4 for **CertRepMessage** syntax.

8.4.11 Key Update Announcement content

When a CA updates its own key pair, the following data structure may be used to announce this event.

```
CAKeyUpdAnnContent ::= SEQUENCE {
    oldWithNew   Certificate, -- old pub signed with new priv
    newWithOld   Certificate, -- new pub signed with old priv
    newWithNew   Certificate  -- new pub signed with new priv
}
```

8.4.12 Certificate Announcement

This structure may be used to announce the existence of certificates.

Note that this message is intended to be used for those cases (if any) where there is no pre-existing method for publication of certificates; it is not intended to be used where, for example, X.500 is the method for publication of certificates.

```
CertAnnContent ::= Certificate
```

8.4.13 Revocation Announcement

When a CA has revoked, or is about to revoke, a particular certificate, it may issue an announcement of this (possibly upcoming) event.

```
RevAnnContent ::= SEQUENCE {
    status            PKIStatus,
    certId            CertId,
    willBeRevokedAt   GeneralizedTime,
    badSinceDate      GeneralizedTime,
    crlDetails        Extensions  OPTIONAL
    -- extra CRL details(e.g. CRL number, reason, location, etc.)
}
```

A CA may use such an announcement to warn (or notify) a subject that its certificate is about to be (or has been) revoked. This would typically be used where the request for revocation did not come from the subject concerned.

The **willBeRevokedAt** field contains the time at which a new entry will be added to the relevant CRLs.

8.4.14 CRL Announcement

When a CA issues a new CRL (or set of CRLs) the following data structure may be used to announce this event.

```
CRLAnnContent ::= SEQUENCE OF CertificateList
```

8.4.15 PKI Confirmation content

This data structure is used in three-way protocols as the final **PKIMessage**. Its content is the same in all cases. Actually there is no content since the **PKIHeader** carries all the required information.

```
PKIConfirmContent ::= NULL
```

8.4.16 PKI General Message content

```
InfoTypeAndValue ::= SEQUENCE {
    infoType       TYPE-IDENTIFIER.&id({InfoTable}),
    infoValue      TYPE-IDENTIFIER.&Type ({InfoTable}{@infoType}) OPTIONAL
}
```

```
    -- Example InfoTypeAndValue contents include, but are not limited to:
    --  { CAProtEncCert    = {id-it 1}, Certificate                        }
    --  { SignKeyPairTypes = {id-it 2}, SEQUENCE OF AlgorithmIdentifier }
    --  { EncKeyPairTypes  = {id-it 3}, SEQUENCE OF AlgorithmIdentifier }
    --  { PreferredSymmAlg = {id-it 4}, AlgorithmIdentifier            }
    --  { CAKeyUpdateInfo  = {id-it 5}, CAKeyUpdAnnContent              }
    --  { CurrentCRL       = {id-it 6}, CertificateList                 }
    -- where {id-it} = {id-pkix 4} = {1 3 6 1 5 5 7 4}
    -- This construct may also be used to define new PKIX Certificate
    -- Management Protocol request and response messages, or general-
    -- purpose (e.g. announcement) messages for future needs or for
    -- specific environments.

GenMsgContent ::= SEQUENCE OF InfoTypeAndValue
    -- May be sent by EE, RA, or CA (depending on message content).
    -- The OPTIONAL infoValue parameter of InfoTypeAndValue will typically
    -- be omitted for some of the examples given above.  The receiver is
    -- free to ignore any contained OBJECT IDs that it does not recognize.
    -- If sent from EE to CA, the empty set indicates that the CA may send
    -- any/all information that it wishes.
```

8.4.17 PKI General Response content

```
GenRepContent ::= SEQUENCE OF InfoTypeAndValue
-- The receiver is free to ignore any contained OBJECT IDs that it does
-- not recognize.
```

8.4.18 Error Message content

```
ErrorMsgContent ::= SEQUENCE {
    pKIStatusInfo          PKIStatusInfo,
    errorCode              INTEGER        OPTIONAL,
    -- implementation-specific error codes
    errorDetails           PKIFreeText    OPTIONAL
    -- implementation-specific error details
}
```

8.5 Transport protocols

No specific transport protocols are mandated for end entities, RAs and CAs to pass PKI messages between them. There is no requirement for specific security mechanisms to be applied at this level if the PKI messages are suitably protected (that is, if the optional PKIProtection parameter is used as specified for each message).

PKI messages may be transported in files containing only the DER encoding of one PKI message, i.e. without extraneous header or trailer information in the file. Such files can be used to transport PKI messages using, e.g. FTP. These files may also be attached to emails or transferred via HTTP (special MIME objects might be defined for this).

8.6 Complete ASN.1 Module

8.6.1 Module Specific for Certification Request Message Format (CRMF)

```
CRMF DEFINITIONS IMPLICIT TAGS ::=
BEGIN
-- EXPORTS ALL; --
IMPORTS
-- Directory Information Framework (X.501)
        Name
          FROM InformationFramework {
            joint-iso-itu-t(2) ds(5) module(1) informationFramework(1) 3 }

     -- Directory Authentication Framework (X.509)
        AlgorithmIdentifier, Extensions, SubjectPublicKeyInfo, Time,
          Version
          FROM AuthenticationFramework {
            joint-iso-itu-t(2) ds(5) module(1) authenticationFramework(7) 3 }
```

```
    -- Directory Selected Attributes (X.520)
       UniqueIdentifier
         FROM SelectedAttributeTypes {
           joint-iso-itu-t(2) ds(5) module(1) selectedAttributeTypes(5) 3 }

    -- Certificate Extensions (X.509)
       GeneralName
         FROM CertificateExtensions {joint-iso-itu-t(2) ds(5)
           module(1) certificateExtensions(26) 0}

    -- Cryptographic Message Syntax
       EnvelopedData
         FROM CryptographicMessageSyntax { iso(1) member-body(2)
           us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
            modules(0) cms(1) };

CertReqMessages ::= SEQUENCE SIZE (1..MAX) OF CertReqMsg

CertReqMsg ::= SEQUENCE {
   certReq   CertRequest,
   pop       ProofOfPossession  OPTIONAL,
   -- content depends upon key type
   regInfo   SEQUENCE SIZE(1..MAX) OF AttributeTypeAndValue OPTIONAL }

CertRequest ::= SEQUENCE {
   certReqId    INTEGER,                      -- ID for matching request and response
certTemplate  CertTemplate,                  -- Selected fields of certificate to be issued
   controls     Controls OPTIONAL }          -- Attributes affecting issuance

CertTemplate ::= SEQUENCE {
   version         [0] Version                  OPTIONAL,
   serialNumber    [1] INTEGER                  OPTIONAL,
   signingAlg      [2] AlgorithmIdentifier      OPTIONAL,
   issuer          [3] EXPLICIT Name            OPTIONAL,
   validity        [4] OptionalValidity         OPTIONAL,
   subject         [5] EXPLICIT Name            OPTIONAL,
   publicKey       [6] SubjectPublicKeyInfo     OPTIONAL,
   issuerUID       [7] UniqueIdentifier         OPTIONAL,
   subjectUID      [8] UniqueIdentifier         OPTIONAL,
   extensions      [9] Extensions               OPTIONAL }

OptionalValidity ::= SEQUENCE {
   notBefore       [0] EXPLICIT Time OPTIONAL,
   notAfter        [1] EXPLICIT Time OPTIONAL } --at least one SHALL be present

Controls  ::= SEQUENCE SIZE(1..MAX) OF AttributeTypeAndValue

AttributeTypeAndValue ::= SEQUENCE {
   type            TYPE-IDENTIFIER.&id ({CRMF-Table}),
   value           TYPE-IDENTIFIER.&Type ({CRMF-Table}{@type})
}

CRMF-Table TYPE-IDENTIFIER::={ ... }

ProofOfPossession ::= CHOICE {
   raVerified      [0] NULL,
   -- used if the RA has already verified that the requester is in
   -- possession of the private key
   signature       [1] POPOSigningKey }

POPOSigningKey ::= SEQUENCE {
   poposkInput           [0] POPOSigningKeyInput OPTIONAL,
   algorithmIdentifier   AlgorithmIdentifier,
   signature             BIT STRING }
   -- The signature (using "algorithmIdentifier") is on the
   -- DER-encoded value of poposkInput.  NOTE – If the CertReqMsg
   -- certReq CertTemplate contains the subject and publicKey values,
   -- then poposkInput SHALL be omitted and the signature SHALL be
   -- computed on the DER-encoded value of CertReqMsg certReq.  If
   -- the CertReqMsg certReq CertTemplate does not contain the public
   -- key and subject values, then poposkInput SHALL be present and
   -- SHALL be signed.  This strategy ensures that the public key is
   -- not present in both the poposkInput and CertReqMsg certReq
   -- CertTemplate fields.

POPOSigningKeyInput ::= SEQUENCE {
   authInfo            CHOICE {
       sender          [0] EXPLICIT GeneralName,
       -- used only if an authenticated identity has been
       -- established for the sender (e.g. a DN from a
       -- previously-issued and currently-valid certificate
       publicKeyMAC      PKMACValue },
       -- used if no authenticated GeneralName currently exists for
       -- the sender; publicKeyMAC contains a password-based MAC
       -- on the DER-encoded value of publicKey
   publicKey           SubjectPublicKeyInfo }  -- from CertTemplate

PKMACValue ::= SEQUENCE {
   algId  AlgorithmIdentifier,
   -- algorithm value shall be PasswordBasedMac {1 2 840 113533 7 66 13}
   -- parameter value is PBMParameter
   value  BIT STRING }

PBMParameter ::= SEQUENCE {
     salt               OCTET STRING,
     owf                AlgorithmIdentifier,
     -- AlgorithmIdentifier for a One-Way Function (SHA-1 recommended)
     iterationCount     INTEGER,
     -- number of times the OWF is applied
     mac                AlgorithmIdentifier
     -- the MAC AlgorithmIdentifier (e.g. DES-MAC, Triple-DES-MAC as in PKCS #11,
}   -- or HMAC as in RFC2104, RFC2202)

-- Object identifier assignments --

id-pkix  OBJECT IDENTIFIER  ::= { iso(1) identified-organization(3)
dod(6) internet(1) security(5) mechanisms(5) 7 }

-- arc for Internet X.509 PKI protocols and their components
id-pkip  OBJECT IDENTIFIER ::= { id-pkix 5 }

-- Registration Controls in CRMF
id-regCtrl OBJECT IDENTIFIER ::= { id-pkip 1 }

id-regCtrl-regToken OBJECT IDENTIFIER ::= { id-regCtrl 1 }
--with syntax:
RegToken ::= UTF8String

id-regCtrl-authenticator OBJECT IDENTIFIER ::= { id-regCtrl 2 }
--with syntax:
Authenticator ::= UTF8String

id-regCtrl-pkiPublicationInfo OBJECT IDENTIFIER ::= { id-regCtrl 3 }
--with syntax:
PKIPublicationInfo ::= SEQUENCE {
   action    INTEGER {
             dontPublish (0),
             pleasePublish (1) },
   pubInfos  SEQUENCE SIZE (1..MAX) OF SinglePubInfo OPTIONAL }
   -- pubInfos SHALL not be present if action is "dontPublish"
   -- (if action is "pleasePublish" and pubInfos is omitted,
   -- "dontCare" is assumed)
```

```
SinglePubInfo ::= SEQUENCE {
   pubMethod    INTEGER {
      dontCare    (0),
      x500        (1),
      web         (2),
      ldap        (3) },
   pubLocation  GeneralName OPTIONAL }

EncryptedKey ::= CHOICE {
   encryptedValue       EncryptedValue,
   envelopedData     [0] EnvelopedData }
   -- The encrypted private key SHALL be placed in the envelopedData
   -- encryptedContentInfo encryptedContent OCTET STRING.

EncryptedValue ::= SEQUENCE {
     intendedAlg    [0] AlgorithmIdentifier    OPTIONAL,
     symmAlg        [1] AlgorithmIdentifier    OPTIONAL,
     encSymmKey     [2] BIT STRING             OPTIONAL,
     keyAlg         [3] AlgorithmIdentifier    OPTIONAL,
     valueHint      [4] OCTET STRING           OPTIONAL,
     encValue           BIT STRING
   }

KeyGenParameters ::= OCTET STRING

id-regCtrl-oldCertID          OBJECT IDENTIFIER ::= { id-regCtrl 5 }
--with syntax:
OldCertId ::= CertId

CertId ::= SEQUENCE {
   issuer           GeneralName,
   serialNumber     INTEGER }

id-regCtrl-protocolEncrKey    OBJECT IDENTIFIER ::= { id-regCtrl 6 }
--with syntax:
ProtocolEncrKey ::= SubjectPublicKeyInfo

-- Registration Info in CRMF
id-regInfo OBJECT IDENTIFIER ::= { id-pkip 2 }

id-regInfo-utf8Pairs    OBJECT IDENTIFIER ::= { id-regInfo 1 }
--with syntax
UTF8Pairs ::= UTF8String

id-regInfo-certReq      OBJECT IDENTIFIER ::= { id-regInfo 2 }
--with syntax
CertReq ::= CertRequest

END
```

8.6.2 General Module

```
-- Note that additional syntax appears in the CRMF module above.
GeneralModule DEFINITIONS EXPLICIT TAGS ::=

   BEGIN

   -- EXPORTS ALL --

   IMPORTS

--  InformationFramework (X.501) --

   Attribute, Name
      FROM InformationFramework {
        joint-iso-itu-t ds(5) module(1) informationFramework(1) 3 }
```

```
-- Directory Authentication Framework (X.509) --

  AlgorithmIdentifier, Certificate, CertificateList, Extensions,
  SubjectPublicKeyInfo
    FROM AuthenticationFramework {
      joint-iso-itu-t ds(5) module(1) authenticationFramework(7) 3 }

-- Certificate Extensions (X.509)

  GeneralName, KeyIdentifier, ReasonFlags
    FROM CertificateExtensions {
      joint-iso-itu-t(2) ds(5) module(1) certificateExtensions(26) 0 }

-- X.843 ISO 15945 (CRMF) --

  CertTemplate, PKIPublicationInfo, EncryptedValue, CertId,
    CertReqMessages
        FROM CRMF;

-- CertificationRequest compatible to PKCS#10

  CertificationRequest ::= SEQUENCE {
    certificationRequestInfo  CertificationRequestInfo,
    signatureAlgorithm        AlgorithmIdentifier,
    signature                 BIT STRING
  }

  CertificationRequestInfo ::= SEQUENCE {
    version       INTEGER,
    subject       Name,
    subjectPKInfo  SubjectPublicKeyInfo,
    attributes    [0] IMPLICIT Attributes
  }

  Attributes ::= SET SIZE(0..MAX) OF Attribute

                    --  Locally defined OIDs  ---- Note that tagging is EXPLICIT in this module.

  PKIMessage ::= SEQUENCE {
    header          PKIHeader,
    body            PKIBody,
    protection      [0] PKIProtection OPTIONAL,
    extraCerts      [1] SEQUENCE SIZE (1..MAX) OF Certificate OPTIONAL
  }

  PKIHeader ::= SEQUENCE {
    pvno            INTEGER   { version1 (0) },
    sender          GeneralName,
    -- identifies the sender
    recipient       GeneralName,
    -- identifies the intended recipient
    messageTime   [0] GeneralizedTime       OPTIONAL,
    -- time of production of this message (used when sender
    -- believes that the transport will be "suitable"; i.e.
    -- that the time will still be meaningful upon receipt)
    protectionAlg  [1] AlgorithmIdentifier  OPTIONAL,
    -- algorithm used for calculation of protection bits
    senderKID      [2] KeyIdentifier        OPTIONAL,
    recipKID       [3] KeyIdentifier        OPTIONAL,
    -- to identify specific keys used for protection
    transactionID  [4] OCTET STRING         OPTIONAL,
    -- identifies the transaction; i.e. this will be the same in
    -- corresponding request, response and confirmation messages
    senderNonce    [5] OCTET STRING         OPTIONAL,
    recipNonce     [6] OCTET STRING         OPTIONAL,
    -- nonces used to provide replay protection, senderNonce
    -- is inserted by the creator of this message; recipNonce
    -- is a nonce previously inserted in a related message by
    -- the intended recipient of this message
    freeText        [7] PKIFreeText          OPTIONAL,
    -- this may be used to indicate context-specific instructions
    -- (this field is intended for human consumption)
    generalInfo     [8] SEQUENCE SIZE (1..MAX) OF
                    InfoTypeAndValue         OPTIONAL
    -- this may be used to convey context-specific information
    -- (this field not primarily intended for human consumption)
  }

  PKIFreeText ::= SEQUENCE SIZE (1..MAX) OF UTF8String
    -- text encoded as UTF-8 String (NOTE – each UTF8String should
    -- include an RFC 1766 language tag to indicate the language
    -- of the contained text)

PKIBody ::= CHOICE {                         -- message-specific body elements
    ir       [0]  CertReqMessages,           --Initialization Request
    ip       [1]  CertRepMessage,            --Initialization Response
    cr       [2]  CertReqMessages,           --Certification Request
    cp       [3]  CertRepMessage,            --Certification Response
    p10cr    [4]  CertificationRequest,      --for compatibility with [PKCS#10]
    kur      [7]  CertReqMessages,           --Key Update Request
    kup      [8]  CertRepMessage,            --Key Update Response
    rr       [11] RevReqContent,             --Revocation Requestrp
    rp       [12] RevRepContent,             --Revocation Response
    ccr      [13] CertReqMessages,           --Cross-Cert. Request
    ccp      [14] CertRepMessage,            --Cross-Cert. Response
    ckuann   [15] CAKeyUpdAnnContent,        --CA Key Update Ann.
    cann     [16] CertAnnContent,            --Certificate Ann.
    rann     [17] RevAnnContent,             --Revocation Ann.
    crlann   [18] CRLAnnContent,             --CRL Announcement
    conf     [19] PKIConfirmContent,         --Confirmation nested
    nested   [20] NestedMessageContent,      --Nested Message
    genm     [21] GenMsgContent,             --General Message
    genp     [22] GenRepContent,             --General Response
    error    [23] ErrorMsgContent            --Error Message
  }

  PKIProtection ::= BIT STRING

  ProtectedPart ::= SEQUENCE {
    header   PKIHeader,
    body     PKIBody
  }

  PasswordBasedMac ::= OBJECT IDENTIFIER --{1 2 840 113533 7 66 13}

  PBMParameter ::= SEQUENCE {
    salt              OCTET STRING,
    owf               AlgorithmIdentifier,
    -- AlgorithmIdentifier for a One-Way Function (SHA-1 recommended)
    iterationCount      INTEGER,
    -- number of times the OWF is applied
    mac               AlgorithmIdentifier
    -- the MAC AlgorithmIdentifier (e.g. DES-MAC, Triple-DES-MAC as in PKCS #11,
  } -- or HMAC as in RFC2104, RFC2202)

  DHBasedMac ::= OBJECT IDENTIFIER --{1 2 840 113533 7 66 30}

  DHBMParameter ::= SEQUENCE {
    owf               AlgorithmIdentifier,
    -- AlgorithmIdentifier for a One-Way Function (SHA-1 recommended)
    mac               AlgorithmIdentifier
    -- the MAC AlgorithmIdentifier (e.g, DES-MAC, Triple-DES-MAC as in PKCS #11,
  } -- or HMAC RFC2104, RFC2202)
```

```
NestedMessageContent ::= PKIMessage

PKIStatus ::= INTEGER {
  granted                 (0),
  -- you got exactly what you asked for
  grantedWithMods         (1),
  -- you got something like what you asked for; the
  -- requester is responsible for ascertaining the differences
  rejection               (2),
  -- you don't get it, more information elsewhere in the message
  waiting                 (3),
  -- the request body part has not yet been processed,
  -- expect to hear more later
  revocationWarning       (4),
  -- this message contains a warning that a revocation is
  -- imminent
  revocationNotification  (5),
  -- notification that a revocation has occurred
  keyUpdateWarning        (6)
  -- update already done for the oldCertId specified in
  -- CertReqMsg
}

PKIFailureInfo ::= BIT STRING {
  -- since a request can fail in more than one way!
  -- More codes may be added in the future if/when required.
  badAlg                  (0),
  -- unrecognized or unsupported Algorithm Identifier
  badMessageCheck         (1),
  -- integrity check failed (e.g. signature did not verify)
  badRequest              (2),
  -- transaction not permitted or supported
  badTime                 (3),
  -- messageTime was not sufficiently close to the system time,
  -- as defined by local policy
  badCertId               (4),
  -- no certificate could be found matching the provided criteria
  badDataFormat           (5),
  -- the data submitted has the wrong format
  wrongAuthority          (6),
  -- the authority indicated in the request is different from the
  -- one creating the response token
  incorrectData           (7),
  -- the requester's data is incorrect (for notary services)
  missingTimeStamp        (8)
  -- when the timestamp is missing but should be there (by policy)
}

PKIStatusInfo ::= SEQUENCE {
  status        PKIStatus,
  statusString  PKIFreeText     OPTIONAL,
  failInfo      PKIFailureInfo  OPTIONAL
}

OOBCert ::= Certificate

OOBCertHash ::= SEQUENCE {
  hashAlg      [0] AlgorithmIdentifier   OPTIONAL,
  certId       [1] CertId                OPTIONAL,
  hashVal          BIT STRING
-- hashVal is calculated over DER encoding of the
-- subjectPublicKey field of the corresponding cert.
}

CertRepMessage ::= SEQUENCE {
  caPubs       [1] SEQUENCE SIZE (1..MAX) OF Certificate OPTIONAL,
  response         SEQUENCE OF CertResponse
}
```

```
CertResponse ::= SEQUENCE {
  certReqId          INTEGER,
  -- to match this response with corresponding request (a value
  -- of -1 is to be used if certReqId is not specified in the
  -- corresponding request)
  status                 PKIStatusInfo,
  certifiedKeyPair   CertifiedKeyPair   OPTIONAL,
  rspInfo                OCTET STRING   OPTIONAL
  -- analogous to the id-regInfo-asciiPairs OCTET STRING defined
  -- for regInfo in CertReqMsg
}

CertifiedKeyPair ::= SEQUENCE {
  certOrEncCert       CertOrEncCert,
  privateKey        [0] EncryptedValue        OPTIONAL,
  publicationInfo  [1] PKIPublicationInfo   OPTIONAL
}

CertOrEncCert ::= CHOICE {
  certificate         [0] Certificate,
  encryptedCert  [1] EncryptedValue
}

KeyRecRepContent ::= SEQUENCE {
  status                  PKIStatusInfo,
  newSigCert    [0]  Certificate                  OPTIONAL,
  caCerts          [1]  SEQUENCE SIZE (1..MAX) OF
                           Certificate                  OPTIONAL,
  keyPairHist    [2]  SEQUENCE SIZE (1..MAX) OF
                           CertifiedKeyPair        OPTIONAL
}

RevReqContent ::= SEQUENCE OF RevDetails

RevDetails ::= SEQUENCE {
  certDetails              CertTemplate,
  -- allows requester to specify as much as they can about
  -- the cert. for which revocation is requested
  -- (e.g. for cases in which serialNumber is not available)
  revocationReason    ReasonFlags          OPTIONAL,
  -- the reason that revocation is requested
  badSinceDate           GeneralizedTime   OPTIONAL,
  -- indicates best knowledge of sender
  crlEntryDetails        Extensions            OPTIONAL
  -- requested crlEntryExtensions
}

RevRepContent ::= SEQUENCE {
  status       SEQUENCE SIZE (1..MAX) OF PKIStatusInfo,
  -- in same order as was sent in RevReqContent
  revCerts   [0] SEQUENCE SIZE (1..MAX) OF CertId OPTIONAL,
  -- IDs for which revocation was requested (same order as status)
  crls          [1] SEQUENCE SIZE (1..MAX) OF CertificateList  OPTIONAL
  -- the resulting CRLs (there may be more than one)
}

CAKeyUpdAnnContent ::= SEQUENCE {
  oldWithNew        Certificate, -- old pub signed with new priv
  newWithOld        Certificate, -- new pub signed with old priv
  newWithNew       Certificate  -- new pub signed with new priv
}

CertAnnContent ::= Certificate

RevAnnContent ::= SEQUENCE {
  status          PKIStatus,
  certId          CertId,
    willBeRevokedAt    GeneralizedTime,
    badSinceDate       GeneralizedTime,
    crlDetails    Extensions  OPTIONAL
    -- extra CRL details(e.g. CRL number, reason, location, etc.)
  }

  CRLAnnContent ::= SEQUENCE OF CertificateList

  PKIConfirmContent ::= NULL

  InfoTypeAndValue ::= SEQUENCE {
    infoType           TYPE-IDENTIFIER.&id ({InfoTable}),
    infoValue          TYPE-IDENTIFIER.&Type ({InfoTable}@infoType}) OPTIONAL
  }

  InfoTable TYPE-IDENTIFIER ::= { ... }

  -- Example InfoTypeAndValue contents include, but are not limited to:
  --  { CAProtEncCert    = {id-it 1}, Certificate                    }
  --  { SignKeyPairTypes = {id-it 2}, SEQUENCE OF AlgorithmIdentifier }
  --  { EncKeyPairTypes  = {id-it 3}, SEQUENCE OF AlgorithmIdentifier }
  --  { PreferredSymmAlg = {id-it 4}, AlgorithmIdentifier            }
  --  { CAKeyUpdateInfo  = {id-it 5}, CAKeyUpdAnnContent             }
  --  { CurrentCRL       = {id-it 6}, CertificateList                }
  -- where {id-it} = {id-pkix 4} = {1 3 6 1 5 5 7 4}
  -- This construct may also be used to define new PKIX Certificate
  -- Management Protocol request and response messages, or general-
  -- purpose (e.g. announcement) messages for future needs or for
  -- specific environments.

  GenMsgContent ::= SEQUENCE OF InfoTypeAndValue
  -- May be sent by EE, RA, or CA (depending on message content).
  -- The OPTIONAL infoValue parameter of InfoTypeAndValue will typically
  -- be omitted for some of the examples given above.  The receiver is
  -- free to ignore any contained OBJECT IDs that it does not recognize.
  -- If sent from EE to CA, the empty set indicates that the CA may send
  -- any/all information that it wishes.

  GenRepContent ::= SEQUENCE OF InfoTypeAndValue
  -- The receiver is free to ignore any contained OBJECT IDs that it does
  -- not recognize.

  ErrorMsgContent ::= SEQUENCE {
    pKIStatusInfo        PKIStatusInfo,
    errorCode          INTEGER          OPTIONAL,
    -- implementation-specific error codes
    errorDetails       PKIFreeText      OPTIONAL
    -- implementation-specific error details
  }


END
```

9 Online Certificate Status Protocol

This clause specifies a protocol useful in determining the current status of a certificate without requiring CRLs. The data structures will be specified in accordance with the RFC document RFC 2560. An overview of the protocol is provided in 9.1. Functional requirements are specified in 9.2. Details of the protocol are in 9.3. Subclause 9.4 contains the ASN.1 module for the data structures needed for the protocol.

NOTE – The ASN.1 code is equivalent to that in the RFC mentioned above though the syntax looks different in parts.

9.1 Protocol Overview

The Online Certificate Status Protocol (OCSP) enables applications to determine the state of an identified certificate. OCSP may be used to satisfy some of the operational requirements of providing more timely revocation information than is possible with CRLs and may also be used to obtain additional status information. An end entity using OCSP services issues a status request to an OCSP TTP and suspends acceptance of the certificate in question until the TTP provides a response.

This protocol specifies the data that needs to be exchanged between an end entity checking the status of a certificate and the TTP providing that status.

9.1.1 Request

An OCSP request contains the following data:

- protocol version;
- service request;
- target certificate identifier;
- optional extensions which may be processed by the OCSP TTP.

Upon receipt of a request, an OCSP TTP determines if:

1) the message is well formed;
2) the TTP is configured to provide the requested service; and
3) the request contains the information needed by the TTP.

If any one of the prior conditions are not met, the OCSP TTP produces an error message; otherwise, it returns a definitive response.

9.1.2 Response

OCSP responses can be of various types. An OCSP response consists of a response type and the bytes of the actual response. There is one basic type of OCSP response that shall be supported by all TTPs providing, and by all entities using, OCSP services. The rest of this subclause pertains only to this basic response type.

All definitive response messages shall be digitally signed. The key used to sign the response shall belong to one of the following:

- the CA who issued the certificate in question;
- a TTP whose public key is trusted by the requester;
- a CA Designated TTP (Authorized TTP) who holds a special certificate issued by the CA indicating that it may issue OCSP responses for that CA.

A definitive response message is composed of:

- version of the response syntax;
- name of the TTP;
- responses for each of the certificates in a request;
- optional extensions;
- signature algorithm OID;
- signature computed across hash of the response.

The response for each of the certificates in a request consists of:

- target certificate identifier;
- certificate status value;
- response validity interval;
- optional extensions.

This Recommendation | International Standard defines the following definitive response indicators for use in the certificate status value:

- **good;**
- **revoked;**
- **unknown.**

The "**good**" state indicates a positive response to the status inquiry. At a minimum, this positive response indicates that the certificate is not revoked, but does not necessarily mean that the certificate was ever issued or that the time at which the response was produced is within the certificate's validity interval. Response extensions may be used to convey additional information on assertions made by the TTP regarding the status of the certificate such as positive statement about issuance, validity, etc.

The "**revoked**" state indicates that the certificate has been revoked (either permanently or temporarily (on hold)).

The "**unknown**" state indicates that the TTP does not know about the certificate being requested.

9.1.3 Exception Cases

In case of errors, the OCSP TTP may return an error message. These messages are not signed. Errors can be of the following types:

- **malformedRequest;**
- **internalError;**
- **tryLater;**
- **sigRequired;**
- **unauthorized.**

A TTP produces the "**malformedRequest**" response if the request received does not conform to the OCSP syntax.

The response "**internalError**" indicates that the OCSP TTP reached an inconsistent internal state. The query should be retried, potentially with another TTP.

In the event that the OCSP TTP is operational, but unable to return a status for the requested certificate, the "**tryLater**" response can be used to indicate that the service exists, but is temporarily unable to respond.

The response "**sigRequired**" is returned in cases where the TTP requires the end entity sign the request in order to construct a response.

The response "**unauthorized**" is returned in cases where the end entity is not authorized to make this query to this TTP.

9.1.4 Semantics of thisUpdate, nextUpdate and producedAt

Responses can contain three times in them: **thisUpdate**, **nextUpdate** and **producedAt**. The semantics of these fields are:

- **thisUpdate**: The time at which the status being indicated is known to be correct;
- **nextUpdate**: The time at or before which newer information will be available about the status of the certificate;
- **producedAt**: The time at which the OCSP TTP signed this response.

If **nextUpdate** is not set, the TTP is indicating that newer revocation information is available all the time.

9.1.5 Response Pre-production

OCSP TTPs may pre-produce signed responses specifying the status of certificates at a specified time. The time at which the status was known to be correct shall be reflected in the **thisUpdate** field of the response. The time at or before which newer information will be available is reflected in the **nextUpdate** field, while the time at which the response was produced will appear in the **producedAt** field of the response.

9.1.6 OCSP Signature Authority Delegation

The key that signs a certificate's status information need not be the same key that signed the certificate. A certificate's issuer explicitly delegates OCSP signing authority by issuing a certificate containing a unique value for **extendedKeyUsage** in the OCSP signer's certificate.

9.1.7 CA Key Compromise

If an OCSP TTP knows that a particular CA's private key has been compromised, it may return the revoked state for all certificates issued by that CA.

9.2 Functional Requirements

9.2.1 Certificate Content

In order to convey to end entities a well-known point of information access, CAs shall provide the capability to include the **AuthorityInfoAccess** extension (defined in RFC 2459, section 4.2.2.1) in certificates that can be checked using OCSP. Alternatively, the **accessLocation** for the OCSP provider may be configured locally at the end entities equipment.

CAs that support an OCSP service, either hosted locally or provided by an Authorized TTP, may provide a value for a **uniformResourceIndicator** (URI) **accessLocation** and the OID value **id-ad-ocsp** for the **accessMethod** in the **AccessDescription SEQUENCE**.

The value of the **accessLocation** field in the subject certificate defines the transport mechanism (e.g. HTTP) used to access the OCSP TTP and may contain other transport dependent information (e.g. a URL).

9.2.2 Signed Response Acceptance Requirements

Prior to accepting a signed response as valid, end entities shall confirm that:

1) The certificate identified in a received response corresponds to that which was identified in the corresponding request;
2) The signature on the response is valid;
3) The identity of the signer matches the intended recipient of the request;
4) The signer is currently authorized to sign the response;
5) The time at which the status being indicated is known to be correct (thisUpdate) is sufficiently recent;
6) When available, the time at or before which newer information will be available about the status of the certificate (nextUpdate) is greater than the current time.

9.3 Detailed Protocol

The ASN.1 syntax imports terms defined in RFC 2459. For signature calculation, the data to be signed is encoded using the ASN.1 distinguished encoding rules (DER).

ASN.1 **EXPLICIT** tagging is used as a default unless specified otherwise.

The terms imported from elsewhere are: **Extensions**, **CertificateSerialNumber**, **SubjectPublicKeyInfo**, **Name**, **AlgorithmIdentifier**, **CRLReason**

9.3.1 Requests

This subclause specifies the ASN.1 specification for a confirmation request.

9.3.1.1 Request Syntax

```
OCSPRequest     ::=     SEQUENCE {
        tbsRequest              TBSRequest,
        optionalSignature       [0]  Signature OPTIONAL }

TBSRequest      ::=     SEQUENCE {
        version                 [0]  Version DEFAULT v1,
        requestorName           [1]  GeneralName OPTIONAL,
        requestList                  SEQUENCE OF Request,
        requestExtensions       [2]  Extensions OPTIONAL }

Signature       ::=SEQUENCE {
        signatureAlgorithm      AlgorithmIdentifier,
        signature               BIT STRING,
        certs                   [0]  SEQUENCE OF Certificate
        OPTIONAL}

Version         ::=     INTEGER  {  v1(0) }

Request         ::=SEQUENCE {
        reqCert                 CertID,
        singleRequestExtensions         [0]  Extensions OPTIONAL }

CertID          ::=SEQUENCE {
        hashAlgorithm           AlgorithmIdentifier,
        issuerNameHash          OCTET STRING, -- Hash of Issuer's DN
        issuerKeyHash           OCTET STRING, -- Hash of Issuer's public key
        serialNumber            CertificateSerialNumber }
```

issuerNameHash is the hash of the Issuer's distinguished name. The hash shall be calculated over the DER encoding of the issuer's name field in the certificate being checked. **issuerKeyHash** is the hash of the Issuer's public key. The hash shall be calculated over the value (excluding tag and length) of the subject public key field in the issuer's certificate. The hash algorithm used for both these hashes, is identified in **hashAlgorithm**.

9.3.1.2 Notes on the Request Syntax

The primary reason to use both the name and the public key to identify the issuer is that it is possible that two CAs may choose to use the same Name (uniqueness in the Name is a recommendation that cannot be enforced). Two CAs will never, however, have the same public key unless the CAs either explicitly decided to share their private key, or the key of one of the CAs was compromised.

Support for any specific extension is optional. The critical flag should not be set for any of them. Subclause 9.3.4 suggests several useful extensions. Unrecognized extensions shall be ignored (unless they have the critical flag set and are not understood).

The requestor may choose to sign the OCSP request. In that case, the signature is computed over the **tbsRequest** structure. If the request is signed, the requestor shall specify its name in the **requestorName** field. Also, for signed requests, the requestor may include certificates that help the OCSP TTP verify the requestor's signature in the certs field of Signature.

9.3.2 Response Syntax

This subclause specifies the ASN.1 specification for a confirmation response.

9.3.2.1 ASN.1 Specification of the OCSP Response

An OCSP response at a minimum consists of a **responseStatus** field indicating the processing status of the prior request. If the value of **responseStatus** is one of the error conditions, **responseBytes** are not set.

```
OCSPResponse ::= SEQUENCE {
        responseStatus OCSPResponseStatus,
        responseBytes[0]  ResponseBytes OPTIONAL }

OCSPResponseStatus ::= ENUMERATED {
        successful          (0),  --Response has valid confirmations
        malformedRequest    (1),  --Illegal confirmation request
        internalError       (2),  --Internal error in issuer
        tryLater            (3),  --Try again later
        --(4) is not used
        sigRequired         (5),  --Must sign the request
        unauthorized        (6)  --Request unauthorized
}
```

The value for responseBytes consists of an **OBJECT IDENTIFIER** and a response syntax identified by that OID encoded as an **OCTET STRING**:

```
ResponseBytes ::=SEQUENCE {
        responseType   OBJECT IDENTIFIER,
        response       OCTET STRING }
```

For a basic OCSP TTP, **responseType** will be **id-pkix-ocsp-basic**:

```
id-pkix-ocsp          OBJECT IDENTIFIER ::= { id-ad-ocsp }
id-pkix-ocsp-basic    OBJECT IDENTIFIER ::= { id-pkix-ocsp 1 }
```

OCSP TTPs shall be capable of responding with responses of the **id-pkix-ocsp-basic** response type. Correspondingly, End entities shall be capable of receiving and processing responses of the **id-pkix-ocsp-basic** response type.

The value for response shall be the DER encoding of **BasicOCSPResponse**:

```
BasicOCSPResponse ::= SEQUENCE {
        tbsResponseData         ResponseData,
        signatureAlgorithm      AlgorithmIdentifier,
        signature               BIT STRING,
        certs                   [0]  SEQUENCE OF Certificate OPTIONAL
}
```

The value for signature shall be computed on the hash of the DER encoding of the **ResponseData**.

```
ResponseData ::= SEQUENCE {
        version              [0] Version DEFAULT v1,
        responderID              ResponderID,
        producedAt               GeneralizedTime,
        responses                SEQUENCE OF SingleResponse,
        responseExtensions   [1] Extensions OPTIONAL }

ResponderID ::= CHOICE {
        byName   [1] Name,
        byKey    [2] KeyHash }

KeyHash ::= OCTET STRING -- SHA-1 hash of TTP's public key (excluding the tag and length fields)

SingleResponse ::= SEQUENCE {
        certID CertID,
        certStatus CertStatus,
        thisUpdate GeneralizedTime,
        nextUpdate         [0] GeneralizedTime OPTIONAL,
        singleExtensions   [1] Extensions OPTIONAL }

CertStatus ::= CHOICE {
        good        [0]IMPLICIT NULL,
        revoked     [1]IMPLICIT RevokedInfo,
        unknown     [2]IMPLICIT UnknownInfo }

RevokedInfo ::= SEQUENCE {
        revocationTime GeneralizedTime,
        revocationReason[0] CRLReason OPTIONAL }

UnknownInfo ::= NULL
```

9.3.2.2 Notes on OCSP Responses

9.3.2.2.1 Time

The **thisUpdate** and **nextUpdate** fields define a recommended validity interval. This interval corresponds to the {**thisUpdate**, **nextUpdate**} interval in CRLs. Responses whose **nextUpdate** value is earlier than the local system time value should be considered unreliable.

Responses whose **thisUpdate** time is later than the local system time should be considered unreliable. Responses where the **nextUpdate** value is not set are equivalent to a CRL with no time for **nextUpdate** (see 9.1.4).

The **producedAt** time is the time at which this response was signed.

9.3.2.2.2 Authorized TTPs

The key that signs a certificate's status information need not be the same key that signed the certificate. A certificate's issuer may explicitly delegate OCSP signing authority by issuing a certificate including an **extendedKeyUsage** extension in the OCSP signer's certificate containing the value **id-kp-OCSPSigning**.

id-kp-OCSPSigning OBJECT IDENTIFIER ::= {id-kp 9}

Since an Authorized OCSP TTP provides status information for a CA, end entities need to know how to check that an authorized TTP's certificate has not been revoked. CAs may choose to deal with this problem in one of three ways:

- A CA may specify that an end entity can trust a TTP for the lifetime of the TTP's certificate. The CA does so by including the extension **id-pkix-ocsp-nocheck**. This should be a non-critical extension. The value of the extension should be **NULL**. CAs issuing such a certificate should realize that a compromise of the TTP's key is as serious as the compromise of the CAs key, at least for the validity period of this certificate. CAs may choose to issue this type of certificate with a very short lifetime and renew it frequently.

  id-pkix-ocsp-nocheck OBJECT IDENTIFIER ::= { id-pkix-ocsp 5 }
- A CA may specify how the TTP's certificate be checked for revocation. This can be done using CRL Distribution Points if the check should be done using CRLs or CRL Distribution Points, or Authority Information Access if the check should be done in some other way.
- A CA may choose not to specify any method of revocation checking for the TTP's certificate, in which case, it would be up to the end entity's local security policy to decide whether that certificate should be checked for revocation or not.

9.3.3 Mandatory and Optional Cryptographic Algorithms

End entities that request OCSP services shall be capable of processing responses signed using DSA keys identified by the DSA **sig-alg-oid** specified in 7.2.2 of RFC 2459. End entities should also be capable of processing RSA signatures as specified in 7.2.1 of RFC 2459. OCSP TTPs shall support the SHA.1 hashing algorithm.

9.3.4 Extensions

This subclause defines some standard extensions. Support for all extensions is optional. For each extension, the definition indicates its syntax, processing performed by the OCSP TTP, and any extensions which are included in the corresponding response.

9.3.4.1 Nonce

The nonce cryptographically binds a request and a response to prevent replay attacks. The nonce is included as one of the **requestExtensions** in requests, while in responses it would be included as one of the **responseExtensions**. In both the request and the response, the nonce will be identified by the object identifier **id-pkix-ocsp-nonce**, while the **extnValue** is the value of the nonce.

```
id-pkix-ocsp-nonce     OBJECT IDENTIFIER ::= { id-pkix-ocsp 2 }
```

9.3.4.2 CRL References

It may be desirable for the OCSP TTP to indicate the CRL on which a revoked or **onHold** certificate is found. This can be useful where OCSP is used between repositories, and also as an auditing mechanism. The CRL may be specified by a URL (the URL at which the CRL is available), a number (CRL number) or a time (the time at which the relevant CRL was created). These extensions will be specified as **singleExtensions**. The identifier for this extension will be **id-pkix-ocsp-crl**, while the value will be **CrlID**.

```
id-pkix-ocsp-crl       OBJECT IDENTIFIER ::= { id-pkix-ocsp 3 }

CrlID ::= SEQUENCE {
          crlUrl     [0]  IA5String OPTIONAL,
          crlNum     [1]  INTEGER OPTIONAL,
          crlTime    [2]  GeneralizedTime OPTIONAL }
```

For the choice **crlUrl**, the **IA5String** will specify the URL at which the CRL is available. For **crlNum**, the **INTEGER** will specify the value of the CRL number extension of the relevant CRL. For **crlTime**, the **GeneralizedTime** will indicate the time at which the relevant CRL was issued.

9.3.4.3 Acceptable Response Types

An End entity may wish to specify the kinds of response types it understands. To do so, it should use an extension with the OID **id-pkix-ocsp-response**, and the value **AcceptableResponses**. The OIDs included in **AcceptableResponses** are the OIDs of the various response types this end entity can accept (e.g. **id-pkix-ocsp-basic**).

```
id-pkix-ocsp-response  OBJECT IDENTIFIER ::= { id-pkix-ocsp 4 }

AcceptableResponses ::= SEQUENCE OF OBJECT IDENTIFIER
```

As noted in 9.3.2.1, OCSP TTPs shall be capable of responding with responses of the **id-pkix-ocsp-basic** response type. Correspondingly, end entities shall be capable of receiving and processing responses of the **id-pkix-ocsp-basic** response type.

9.3.4.4 Archive Cutoff

An OCSP TTP may choose to retain revocation information beyond a certificate's expiration. The date obtained by subtracting this retention interval value from the **producedAt** time in a response is defined as the certificate's "archive cutoff" date.

OCSP-enabled applications would use an OCSP archive cutoff date to contribute to a proof that a digital signature was (or was not) reliable on the date it was produced even if the certificate needed to validate the signature has long since expired.

OCSP TTPs that provide support for such historical reference should include an archive cutoff date extension in responses. If included, this value shall be provided as an OCSP **singleResponse** extension identified by **id-pkix-ocsp-archive-cutoff** and of syntax **GeneralizedTime**:

```
id-pkix-ocsp-archive-cutoff  OBJECT IDENTIFIER ::= { id-pkix-ocsp 6 }
```

```
ArchiveCutoff ::= GeneralizedTime
```

To illustrate, if a TTP is operated with a 7-year retention interval policy and status was produced at time t1, then the value for **ArchiveCutoff** in the response would be (t1 – 7 years).

9.3.4.5 CRL Entry Extensions

CRL Entry Extensions are also supported as **singleExtensions**.

9.3.4.6 Service Locator

An OCSP TTP may be operated in a mode whereby the TTP receives a request and routes it to the OCSP TTP which is known to be authoritative for the identified certificate. The **serviceLocator** request extension is defined for this purpose.

```
id-pkix-ocsp-service-locator OBJECT IDENTIFIER ::= { id-pkix-ocsp 7 }

ServiceLocator ::= SEQUENCE {
        issuer   Name,
        locator AuthorityInfoAccessSyntax OPTIONAL }
```

Values for these fields are obtained from the corresponding fields in the subject certificate.

9.4 ASN.1 Module for OCSP

```
OCSP DEFINITIONS EXPLICIT TAGS ::=

BEGIN

IMPORTS
-- Directory Information Framework (X.501) --
   Name
     FROM InformationFramework {
       joint-iso-itu-t ds(5) module(1) informationFramework(1) 3 }
   -- Directory Authentication Framework (X.509) --
   AlgorithmIdentifier, Certificate, CertificateSerialNumber,
   Extensions
     FROM AuthenticationFramework {
       joint-iso-itu-t ds(5) module(1) authenticationFramework(7) 3 }
   -- Directory Certificate Extensions (X.509) --
   CRLReason, GeneralName
     FROM CertificateExtensions {joint-iso-itu-t ds(5)
       module(1) certificateExtensions(26) 0 }
   -- PKIX (RFC 2459) --
   AuthorityInfoAccessSyntax
     FROM PKIX1Implicit93 {
       iso(1) identified-organization(3) dod(6) internet(1)
         security(5) mechanisms(5) pkix(7) id-mod(0)
            id-pkix1-implicit-93(4) }
   id-kp, id-ad-ocsp
     FROM PKIX1Explicit93 {
       iso(1) identified-organization(3) dod(6) internet(1)
         security(5) mechanisms(5) pkix(7) id-mod(0)
            id-pkix1-explicit-93(3) };

OCSPRequest    ::=SEQUENCE {
        tbsRequest      TBSRequest,
        optionalSignature   [0] Signature OPTIONAL }

TBSRequest   ::=SEQUENCE {
        version                 [0] Version DEFAULT v1,
        requestorName           [1] GeneralName OPTIONAL,
        requestList                 SEQUENCE OF Request,
        requestExtensions       [2] Extensions OPTIONAL }
```

```
Signature  ::=SEQUENCE {
            signatureAlgorithm   AlgorithmIdentifier,
            signature BIT STRING,
            certs   [0] SEQUENCE OF Certificate OPTIONAL }

Version  ::=  INTEGER  {  v1(0) }

Request ::=     SEQUENCE {
            reqCert      CertID,
            singleRequestExtensions    [0] Extensions OPTIONAL }

CertID ::= SEQUENCE {
            hashAlgorithm       AlgorithmIdentifier,
            issuerNameHash OCTET STRING, -- Hash of Issuer's DN
            issuerKeyHash OCTET STRING, -- Hash of Issuer's public key
            serialNumber CertificateSerialNumber }

OCSPResponse ::= SEQUENCE {
            responseStatus       OCSPResponseStatus,
            responseBytes        [0] ResponseBytes OPTIONAL }

OCSPResponseStatus ::= ENUMERATED {
            successful(0), --Response has valid confirmations
            malformedRequest(1), --Illegal confirmation request
            internalError(2), --Internal error in issuer
            tryLater(3), --Try again later
            --(4) is not used
            sigRequired(5), --Must sign the request
            unauthorized(6) --Request unauthorized
}

ResponseBytes ::=SEQUENCE {
            responseType   OBJECT IDENTIFIER,
            response          OCTET STRING }

BasicOCSPResponse ::= SEQUENCE {
            tbsResponseData ResponseData,
            signatureAlgorithm AlgorithmIdentifier,
            signature BIT STRING,
            certs   [0] SEQUENCE OF Certificate OPTIONAL
}

ResponseData ::= SEQUENCE {
            version                      [0] Version DEFAULT v1,
            responderID                      ResponderID,
            producedAt                       GeneralizedTime,
            responses                        SEQUENCE OF SingleResponse,
            responseExtensions   [1] Extensions OPTIONAL }

ResponderID ::= CHOICE {
            byName   [1] Name,
            byKey      [2] KeyHash }

KeyHash ::= OCTET STRING --SHA-1 hash of TTP's public key
            --(excluding the tag and length fields)

SingleResponse ::= SEQUENCE {
            certID CertID,
            certStatus CertStatus,
            thisUpdate GeneralizedTime,
            nextUpdate[0] GeneralizedTime OPTIONAL,
            singleExtensions[1] Extensions OPTIONAL }

CertStatus ::= CHOICE {
            good           [0]   IMPLICIT NULL,
            revoked       [1]   IMPLICIT RevokedInfo,
            unknown     [2]   IMPLICIT UnknownInfo }

RevokedInfo ::= SEQUENCE {
            revocationTime GeneralizedTime,
            revocationReason[0] CRLReason OPTIONAL }
```

```
UnknownInfo ::= NULL

ArchiveCutoff ::= GeneralizedTime

AcceptableResponses ::= SEQUENCE OF OBJECT IDENTIFIER

ServiceLocator ::= SEQUENCE {
        issuer    Name,
        locator   AuthorityInfoAccessSyntax }

-- Object Identifiers
id-kp-OCSPSigning               OBJECT IDENTIFIER ::= { id-kp 9 }
id-pkix-ocsp                    OBJECT IDENTIFIER ::= { id-ad-ocsp }
id-pkix-ocsp-basic              OBJECT IDENTIFIER ::= { id-pkix-ocsp 1 }
id-pkix-ocsp-nonce              OBJECT IDENTIFIER ::= { id-pkix-ocsp 2 }
id-pkix-ocsp-crl                OBJECT IDENTIFIER ::= { id-pkix-ocsp 3 }
id-pkix-ocsp-response           OBJECT IDENTIFIER ::= { id-pkix-ocsp 4 }
id-pkix-ocsp-nocheck            OBJECT IDENTIFIER ::= { id-pkix-ocsp 5 }
id-pkix-ocsp-archive-cutoff     OBJECT IDENTIFIER ::= { id-pkix-ocsp 6 }
id-pkix-ocsp-service-locator    OBJECT IDENTIFIER ::= { id-pkix-ocsp 7 }
END
```

Annex A

Interworking

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

The provision of certificate management services may require the interworking of different service providers. If a set of CAs is considered, each of which is running their service within their own realm, then entities need to have assured public keys of CAs from other domains. There are two possible basic models: the first one is a hierarchical one based on certificate chains, in the second one, CAs may cross-certify each other. Hybrid models between these two are possible.

In the first model, authorities are placed hierarchically under a "root"-CA that issues certificates to so called subordinate CAs. These CAs may issue certificates to other CAs below them within this hierarchy, or to entities.

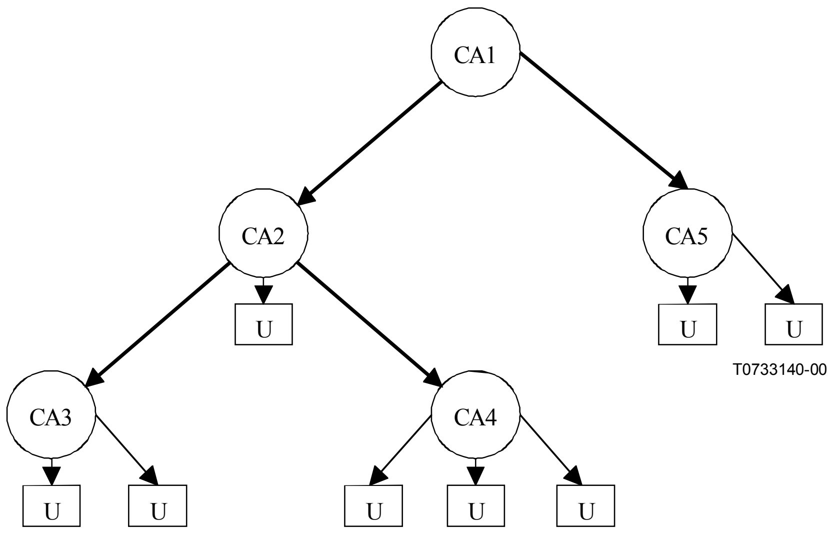

Figure A.1 – Hierarchy of CAs

CA1 is called "root"-CA, see Figure A.1. The duty of the root-CA is to register its subordinate CAs; in Figure A.1 these are CA2 and CA5. The subordinate CAs may register further CAs and/or entities. Each CA should operate in accordance with a common policy so that a common level of trust, or quality of service, can be achieved.

In the second model, independent CAs cross-certify each other resulting in a network of interconnected CAs. Cross-certification is a bilateral agreement between two CAs, where one issues a certificate for the other one, or where both of them issue a certificate for the other one.

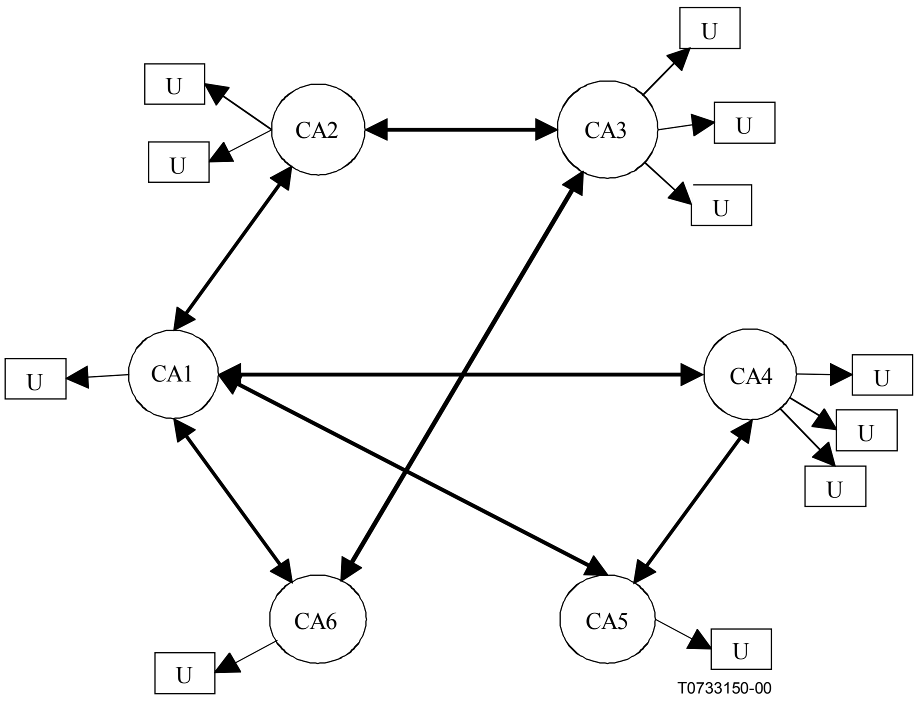

Figure A.2 – Example for a Network of cross-certified CAs

Annex B

Algorithms

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

B.1 Hash Algorithms

The following hash algorithms are suitable for the use in the context of digital signatures:

a) RIPEMD-160:

DOBBERTIN, BOSSELAERS (A.), PRENEEL (B.): *RIPEMD-160: A strengthened version of RIPEMD, Fast Software Encryption,* Cambridge Workshop 1996, LNCS, Band 1039, S. 71-82, Springer-Verlag, 1996.

b) SHA-1:

NIST: FIPS Publication 180-1: *Secure Hash Standard (SHS-1),* May 1995.

Both are described in:

ISO/IEC 10118-3:1998, *Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions.*

B.2 Digital Signature Algorithms

The following digital signature algorithms are suitable in the context of this Recommendation | International Standard.

a) DSA:

NIST: FIPS Publication 186: *Digital Signature Standard (DSS)*, May 1994.

b) RSA:

RIVEST, SHAMIR (A.), ADLEMAN (B): A method for obtaining digital signatures and public key cryptosystems, *Communications of the ACM*, Vol. 21, No. 2, 1978.

c) DSA-like schemes based on Elliptic Curves:

– ISO/IEC 14883-3, Annex A.2.1 ('Elliptic Curve DSA');

– ISO/IEC WD 15946-2;

– IEEE: P1363 Standard (Draft), 6 February 1997, Clause 5.3.3 ('Nyberg-Rueppel version');

– IEEE: P1363 Standard (Draft), 6 February 1997, Clause 5.3.4 ('DSA version').

A number of algorithms are described in:

ISO/IEC 14888-3:1998, *Information technology – Security techniques – Digital signatures with appendix – Part 3: Certificate-based mechanisms.*

Annex C

Bibliography

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

COM(1997)503, *'Ensuring Security and Trust in Electronic Communication', Communication from the Commission to the European Parliament, the Council, the Economic and Social Committee and the Committee of the Regions,* October 1997.

Directive 1999/93/EC of the European Parliament and of the Council on a Communinty framework for electronic signatures of 13 December 1999.

ECBS TR 402, European Committee for Banking Standards, Technical Report 402: Certification Authorities, Vol. 1, 1997.

ETSI EG/SEC-003000 Requirements for Trusted Third Party Services (Edition 1), Version 7.0, July 1997.

FIPS PUB 140-1, *Federal Information Processing Standard Publication 140-1, "Security Requirements for Cryptographic Modules", U.S. Department of commerce, National Institute of Standards and Technology,* January 1994.

ITU-T Recommendation X.511 (1997) | ISO/IEC 9594-3:1998, *Information technology – Open Systems Interconnection – The Directory: Abstract service definition.*

ITU-T Recommendation X.519 (1997) | ISO/IEC 9594-5:1998, *Information technology – Open Systems Interconnection – The Directory: Protocol specifications.*

ITU-T Recommendation X.810 (1995) | ISO/IEC 10181-1:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Overview.*

ITU-T Recommendation X.811 (1995) | ISO/IEC 10181-2:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Authentication framework.*

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

ITU-T Recommendation X.813 (1996) | ISO/IEC 10181-4: 1997, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Non-repudiation framework.*

ITU-T Recommendation X.814 (1995) | ISO/IEC 10181-5:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Confidentiality framework.*

ITU-T Recommendation X.815 (1995) | ISO/IEC 10181-6:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Integrity framework.*

ITU-T Recommendation X.816 (1995) | ISO/IEC 10181-7:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Security audit and alarms framework.*

ISO/DIS 15782-1, *Certificate Management for financial services – Part 1: Public Key Certificates.*

ISO/IEC 10118-1:1994, *Information technology – Security techniques – Hash-functions – Part 1: General.*

ISO/IEC 10118-2:1994, *Information technology – Security techniques – Hash-functions – Part 2: Hash-functions using an n-bit block cipher algorithm.*

ISO/IEC 10118-3:1998, *Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions.*

ISO 9735:1998, *Electronic data interchange for administration, commerce and transport (EDIFACT) – Application level syntax rules.*

ISO/IEC 15408-1:1999, *Information technology – Security techniques – Evaluation criteria for IT security – Part 1: Introduction and general model.*

ISO/IEC 15408-2:1999 *Information technology – Security techniques – Evaluation criteria for IT Security – Part 2: Security functional requirements.*

ISO/IEC 15408-3:1999, *Information technology – Security techniques – Evaluation criteria for IT Security – Part 3: Security assurance requirements.*

ITSEC, *Information Technology Security Evaluation Criteria (ITSEC), Harmonized Criteria of France, Germany, the Netherlands, the United Kingdom, Version 1.2,* June 1992.

Minimum Interoperability Specification for PKI Components (MISPC), NIST Special Publication 800-15, Sept. 1997.

PKCS Papers

PKCS #1, RSA Laboratories, PKCS #1: *RSA Encryption Standard,* Version 1.5, November 1993.

PKCS #10, RSA Laboratories, PKCS #10: *Certification Request Standard*, Version 1.5, November 1993.

PKCS #11, RSA Laboratories, PKCS #11: *Cryptographic Token Interface Standard*, Version 1.0, April 1995.

PKCS #3, RSA Laboratories, PKCS #3: *Diffie-Hellman Key Agreement Standard,* Version 1.4, November 1993.

PKCS #5, RSA Laboratories, PKCS #5: *Password-Based Encryption Standard*, Version 1.5, November 1993.

PKCS #6, RSA *Laboratories*, PKCS #6: *Extended-Certificate Syntax Standard*, Version 1.5, November 1993.

PKCS #7, RSA Laboratories, PKCS #7: *Cryptographic Message Standard*, Version 1.5, November 1993, Extensions and revisions, 1997.

PKCS #8, RSA Laboratories, PKCS #8: *Private Key Information Syntax Standard*, Version 1.5, November 1993.

PKCS #9, RSA Laboratories, PKCS #9: *Selected Attribute Types*, Version 1.5, November 1993.

PKIX Papers

RFC 1421, *Privacy Enhancement for Electronic Mail: Part 1: Message Encryption and Authentication Procedures,* February 1993.

RFC 1422, *Privacy Enhancement for Electronic Mail: Part 2: Certificate-Based Key Management,* February 1993.

RFC 1423, *Privacy Enhancement for Electronic Mail: Part 3: Algorithms, Modes, and Identifiers,* February 1993.

RFC 1424, *Privacy Enhancement for Electronic Mail: Part 4: Key Certification and Related Services,* February 1993.

RFC 1510, *The Kerberos Network Authentication Services,* September 1993.

RFC 1750, *Randomness Recommendations for Security,* December 1994.

RFC 1766, *Tags for the Identification of Languages,* March 1995.

RFC 2104, *HMAC: Keyed-Hashing for Message Authentication,* February 1997.

RFC 2459, *Internet X.509 Public Key Infrastructure Certificate and CRL Profile,* January 1999.

RFC 2510, *Internet X.509 Public Key Infrastructure Certificate Management Protocols,* March 1999.

RFC 2511, *Internet X.509 Public Key Infrastructure Certificate Request Message Format,* March 1999.

RFC 2527, *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework,* March 1999.

RFC 2559, *Internet X.509 Public Key Infrastructure Operational Protocols – LDAPv2,* April 1999.

RFC 2560, *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP,* June 1999.

Time Stamp Protocols, Internet Draft, September 1998 (work in progress); Adams C., Cain P., Pinkas D., Zuccherato R.

SET Secure Electronic Transaction Specification, Book 1: Business Description, Version 1.0, 31 May 1997.

SET Secure Electronic Transaction Specification, Book 2: Programmer's Guide, Version 1.0, 31 May 1997.

SET Secure Electronic Transaction Specification, Book 3: Formal Protocol Definition, Version 1.0, 31 May 1997.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	**Data networks and open system communications**
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	Languages and general software aspects for telecommunication systems

Geneva, 2001