Internet Draft C. Adams (Entrust Technologies)
PKIX Working Group S. Farrell (SSE)
draft-ietf-pkix-ipki3cmp-02.txt
Expires in 6 months June 1997

Internet Public Key Infrastructure
Part III: Certificate Management Protocols

Status of this Memo

Abstract

This is a draft of the Internet Public Key Infrastructure (X.509) Certificate Management Protocols. Protocol messages are defined for all relevant aspects of certificate creation and management.

1. Introduction

The layout of this draft is as follows:

- Section 1 contains an overview of PKI management;
- Section 2 contains discussion of assumptions and restrictions;
- Section 3 contains data structures used for PKI management messages;
- Section 4 defines the functions that are to be carried out in PKI management including those that must be supported by conforming implementations and those that are optional;
- Section 5 describes a simple protocol for transporting PKI messages;
- the Appendices specify profiles for conforming implementations and provide an ASN.1 module containing the syntax for all defined messages.

1.1 PKI Management Overview

The PKI must be structured to be consistent with the types of individuals who must administer it. Providing such administrators with unbounded choices not only complicates the software required but also increases the chances that a subtle mistake by an administrator or software developer will result in broader compromise. Similarly, restricting administrators with cumbersome mechanisms will cause them not to use the PKI.

Management protocols are required to support on-line interactions between Public Key Infrastructure (PKI) components. For example, a management protocol might be used between a CA and a client system with which a key pair is associated, or between two CAs which cross-certify each other.

2.1 PKI Management Model

Before specifying particular message formats and procedures we first define the entities involved in PKI management and their interactions (in terms of the PKI management functions required). We then group these functions in order to accommodate different identifiable types of end entities.

1.2 Definitions of PKI Entities

The entities involved in PKI management include the end entity (i.e. the entity to be named in the subject field of a certificate) and the certification authority (i.e. the entity named in the issuer field of a certificate). A registration authority may also be involved in PKI management.

1.2.1 Subjects and End Entities

The term "subject" is used here to refer to the entity named by the subject field of a certificate; when we wish to distinguish the tools and/or software used by the subject (e.g. a local certificate management module) we will use the term "subject equipment". In general, we prefer the term "end entity" rather than subject in order to avoid confusion with the field name.

It is important to note that the end entities here will include not only human users of applications, but also applications themselves (e.g. for IP security). This factor influences the protocols which the PKI management operations use; e.g., applications software is far more likely to know exactly which certificate extensions are required than are human users. PKI management entities are also end entities in the sense that they are sometimes named in the subject field of a certificate or cross-certificate. Where appropriate, the term "end-entity" will be used to refer to end entities who are not PKI management entities.

All end entities require secure local access to some information -- at a minimum, their own name and private key, the name of a CA which is directly trusted by this subject and that CA's public key (or a fingerprint of the public key where a self-certified version is available elsewhere). Implementations may use secure local storage for more than this minimum (e.g. the end entity's own certificate or application-specific information). The form of storage will also vary -- from files to tamper resistant cryptographic tokens. Such local trusted storage is referred to here as the end entity's Personal Security Environment (PSE).

Though PSE formats are out of scope of this document (they are very dependent on equipment, et cetera), a generic interchange format for PSEs is defined here - a certification response message may be used.

1.2.2 Certification Authority

The certification authority (CA) may or may not actually be a real "third party" from the end entity's point of view. Quite often, the CA will actually belong to the same organisation as the end entities it supports.

Again, we use the term CA to refer to the entity named in the issuer field of a certificate; when it is necessary to distinguish the software or hardware tools used by the CA we use the term "CA equipment".

The CA equipment will often include both an "off-line" component and an "on-line" component, with the CA private key only available to the "off-line" component. This is, however, a matter for implementers (though it is also relevant as a policy issue).

We use the term "root CA" to indicate a CA which is directly trusted by an end entity, that is, securely acquiring the value of a root CA public key requires some out-of-band step(s). This term does not indicate that a root CA is at the top of any hierarchy, simply that the CA in question is trusted directly.

A subordinate CA is one which is not a root CA for the end entity in question. Often, a subordinate CA will not be a root CA for any entity but this is not mandatory.

1.2.3 Registration Authority

In addition to end entities and CAs, many environments call for the existence of a registration authority (RA) separate from the certification authority. The functions which the registration authority may carry out will vary from case to case but may include personal authentication, token distribution, revocation reporting, name assignment, key generation, archival of key pairs, et cetera.

This document views the RA as an optional component - when it is not present the CA is assumed to be able to carry out the RA's functions so that the PKI management protocols are the same from the end entity's point of view.

Again, we distinguish, where necessary, between the RA and the tools used (the "RA equipment").

Note that an RA is itself an end entity. We further assume that all RAs are in fact certified end entities and that RA private keys are usable for signing. How a particular CA equipment identifies some end entities as RAs is an implementation issue (so there is no special RA certification operation). We do not mandate that the RA is certified by the CA with which it is interacting at the moment (so one RA may work with more than one CA whilst only being certified once).

In some circumstances end entities will communicate directly with a CA even where an RA is present. For example, for initial registration and/or certification the subject may use its RA, but communicate directly with the CA in order to refresh its certificate.

1.3 PKI Management Requirements

The protocols given here meet the following requirements on PKI management.

1. **PKI management must conform to ISO 9594-8 and the associated** amendments (certificate extensions)

2. **PKI management must conform to the other parts of this series.**

3. **It must be possible to regularly update any key pair without** affecting any other key pair.

4. **The use of confidentiality in PKI management protocols must be kept** to a minimum in order to ease regulatory problems.

5. **PKI management protocols must allow the use of different industry-**standard cryptographic algorithms, (specifically including, RSA, DSA, MD5, SHA-1) -- this means that any given CA, RA, or end entity may, in principal, use whichever algorithms suit it for its own key pair(s).

6. **PKI management protocols must not preclude the generation of key** pairs by the end entity concerned, by an RA, or by a CA -- key generation may also occur elsewhere, but for the purposes of PKI management we can regard key generation as occurring wherever the key is first present at an end entity, RA or CA.

7. **PKI management protocols must support the publication of certificates** by the end entity concerned, by an RA or by a CA. Different implementations and different environments may choose any of the above approaches.

8. **PKI management protocols must support the production of CRLs by** allowing certified end entities to make requests for the revocation of certificates - this must be done in such a way that the denial-of-service attacks which are possible are not made simpler.

9. **PKI management protocols must be usable over a variety of "transport"** mechanisms, specifically including mail, http, TCP/IP and ftp.

10. **Final authority for certification creation rests with the CA; no RA** or end entity equipment can assume that any certificate issued by a CA will contain what was requested -- a CA may alter certificate field values or may add, delete or alter extensions according to its operating policy; the only exception to this is the public key, which the CA may not modify (assuming that the CA was presented with the public key value). In other words, all PKI entities (end entities, RAs and CAs) must be capable of handling responses to requests for certificates in which the actual certificate issued is different from that requested -- for example, a CA may shorten the validity period requested.

11. **A graceful, scheduled change-over from one non-compromised** CA key pair to the next must be supported (CA key update). An end entity whose PSE contains the new CA public key (following a CA key update) must also be able to verify certificates verifiable using the old public key. End entities who directly trust the old CA key pair must also be able to verify certificates signed using the new CA private key. (Required for situations where the old CA public key is "hardwired" into the end entity's cryptographic equipment).

12. **The Functions of an RA may, in some implementations or** environments, be carried out by the CA itself. The protocols must be designed so that end entities will use the same protocol (but, of course, not the same key!) regardless of whether the communication is with an RA or CA.

13. **Where an end entity requests a certificate containing a given public** key value, the end entity must be ready to demonstrate possession of the corresponding private key value (if this is required by the CA/RA with whom the end entity is communicating). This may be accomplished in various ways, depending on the type of certification request. See the section "Proof of Possession of Private Key" for details

PKI Management Operations

The following diagram shows the relationship between the entities defined above in terms of the PKI management operations. The letters in the diagram indicate "protocols" in the sense that a defined set of PKI management messages can be sent along each of the lettered lines.

```
                cert. publish        +------------+
      +---+           +------------  | End Entity |   Operations
      | C |           | g            +------------+
      | e |  <--------+                | ^      initial
      | r |                          a | | b     registration/
      | t |       PKI "users"          | |       certification
      |   |                            | |      key pair recovery
      | / |                            | |      key pair update
      |   |                            | |      certificate update
      | C |                            V |      revocation request
      | R |             -------+-+-----+-+------+-+-----
      | L |   PKI management   | ^              | ^
      |   |      entities    a | | b          a | | b
      |   |                    V |              | |
      | R |             g   +------+    d       | |
      | e |   <------------ | RA   | <-----+    | |
      | p |      cert.      |      | ----+ |    | |
      | o |       publish   +------+   c | |    | |
      | s |                              | |    | |
      | i |                              V |    V |
      | t |          h                 +------------+   i
      | o |   <------------------------| CA         |------->
      | r |                            +------------+  "out-of-band"
      | y |      cert. publish              | ^         publication
      |   |      CRL publish                | |
      +---+                                 | |    cross-certification
                                          e | | f  cross-certificate
                                            | |       update
                                            | |
                                            V |
                                          +------+
                                          | CA-2 |
                                          +------+
```

Figure 1 - PKI Entities

At a high level the set of operations for which management messages are defined can be grouped as follows.

1 CA establishment: When establishing a new CA, certain steps are
required (e.g., production of initial CRLs, export of CA public

key).

2 End entity initialisation: this includes importing a CA public key and requesting information about the options supported by a PKI management entity.

3 Certification: various operations result in the creation of new certificates:

3.1 initial registration/certification: This is the process whereby a subject first makes itself known to a CA or RA, prior to the CA issuing a certificate or certificates for that user. 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 subject 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 subject equipment must be securely initialized with the public key of a CA, to be used in validating certificate paths. Furthermore, a subject typically needs to be initialized with its own key pair(s).

3.2 key pair update: Every key pair needs to be updated regularly (i.e., replaced with a new key pair), and a new certificate needs to be issued.

3.3 certificate update: As certificates expire they may be "refreshed" if nothing relevant in the environment has changed.

3.4 CA key pair update: As with end entities, CA key pairs need to be updated regularly; however, different mechanisms are required.

3.5 cross-certification: An initiating CA provides to a responding CA the information necessary for the responding CA to issue a cross-certificate. Note: this action may be mutual, so that two cross-certificates are issued (one in each direction).

3.6 cross-certificate update: Similar to a normal certificate update but involving a cross-certificate.

4 Certificate/CRL discovery operations: some PKI management operations result in the publication of certificates or CRLs

4.1 certificate publication: Having gone to the trouble of producing a certificate some means for publishing it is needed. The "means" defined in PKIX may involve the messages specified in Sections 3.3.13 - 3.3.16, or may involve other methods (LDAP, for example) as described in Part II of this series (see [PKIX-2]).

4.2 CRL publication: As for certificates.

5 Recovery operations: some PKI management operations are used when an end entity has "lost" it s PSE

5.1 key pair recovery: As an option, user client key materials (e.g., a user's private key used for decryption purposes) may be backed up by a CA, an RA or a key backup system associated with a CA or RA. If a subject needs to recover these backed up key materials (e.g., as a result of a forgotten password or a lost key chain file), a protocol exchange may be needed to support such recovery.

6 Revocation operations: some PKI operations result in the creation of new CRL entries and/or new CRLs

6.1 revocation request: An authorized person advises a CA of an abnormal situation requiring certificate revocation.

7 PSE operations: whilst the definition of PSE operations (e.g. moving a PSE, changing a PIN, etc.) are beyond the scope of this specification, we do define a PKIMessage which can form the basis of such operations.

Note that on-line protocols are not the only way of implementing the above operations. For all operations there are off-line methods of achieving the same result, and this specification does not mandate use of on-line protocols. For example, when hardware tokens are used, many of the operations may be achieved as part of the physical token delivery.

Later sections define a set of standard protocols supporting the above operations. The protocols for conveying these exchanges in different environments (file based, on-line, E-mail, and WWW) may also be specified.

2. Assumptions and restrictions

2.1 End entity initialisation

The first step for and end entity in dealing with PKI management entities is to request information about the PKI functions supported and optionally to securely acquire a copy of the relevant root CA public key(s).

2.2 Initial registration/certification

There are many schemes which 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.

We can however, classify the initial registration / certification schemes which are supported by this specification. Note that the word "initial", above, is crucial - we are dealing with 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.

Having classified the schemes which are supported by this specification we can then specify some as mandatory and some as optional. The goal is that the mandatory schemes cover a sufficient number of the cases which will arise in real use, whilst the optional schemes are available for special cases which arise less frequently. In this way we achieve a balance between flexibility and ease of implementation.

We will now describe the classification of initial registration / certification schemes.

2.2.1 Criteria used

2.2.1.1 Initiation of registration / certification

In terms of the PKI messages which are produced we can regard the initiation of the initial registration / certification exchanges as occurring wherever the first PKI message relating to the end entity is produced. Note that the real world initiation of the registration / certification procedure may occur elsewhere (e.g. a personnel department may telephone an RA operator).

The possible locations are: at the end entity, an RA or a CA.

2.2.1.2 End entity message origin authentication

The on-line messages produced by the end entity which requires a certificate may be authenticated or not. The requirement here is to authenticate the origin of any messages from the end entity to the PKI (CA/RA).

In this specification, such authentication is achieved by the PKI (CA/RA) issuing the end entity with a secret value (initial authentication key) and reference value (used to identify the transaction) via some out-of-band means. The initial authentication key can then be used to protect relevant PKI messages.

We can thus classify the initial registration/certification scheme according to whether or not the on-line end entity -> PKI messages are authenticated or not.

Note 1: We do not discuss the authentication of the PKI -> end entity messages here as this is always required. In any case, it can be achieved simply once the root-CA public key has been installed at the end entity s equipment or based on the initial authentication key.

Note 2: An initial registration / certification procedure can be secure where the messages from the end entity are authenticated via some out-of-band means (e.g. a subsequent visit).

2.2.1.3 Location of key generation

In this specification, key generation is regarded as occurring wherever either the public or private component of a key pair first occurs in a PKI message. Note that this does not preclude a centralised key generation service - the actual key pair may have been generated elsewhere and transported to the end entity, RA or CA.

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

2.2.1.4 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 the initial authentication key or other means).

This gives two further possibilities: confirmed or not.

2.2.2 Mandatory schemes

The criteria above allow for a large number of initial registration / certification schemes. This specification mandates that conforming RA/CA equipment must support both of the schemes listed below. Conforming end entity equipment must support one of the schemes listed below.

2.2.2.1 Centralised scheme

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

- initiation occurs at the certifying CA;
- no on-line message authentication is required;
- key generation occurs at the certifying CA;
- no confirmation message is required.

In terms of message flow, this scheme means that the only message required is sent from the CA to the end entity. The message must contain the entire PSE for the end entity. Some out-of-band means must be provided to allow the end entity to authenticate the message received.

2.2.2.2 Basic authenticated scheme

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

- initiation occurs at the end entity
- message authentication is required
- key generation occurs at the end entity
- a confirmation message is required

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

```
      End entity                                          CA
      ==========                                      =============
                   out-of-band distribution of
                   initial authentication key and
                   reference value
      Key generation
      Creation of certification request
      Protect request with IAK
                   -->>--certification request-->>
                                                     verify request
                                                     process request
                                                     create response
                   --<<--certification response--<<--
      handle response
      create confirmation
                   -->>--confirmation message-->>--
                                                     verify confirmation
```

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

2.3 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 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 key corresponding to the public key for which a certificate is requested. A given CA/RA is free to choose whether or not to enforce POP in its certification exchanges (i.e., this may be a policy issue). However, it is STRONGLY RECOMMENDED that CAs/RAs enforce POP because there are currently many non-PKIX operational protocols in use (various electronic mail protocols are one example) which do not explicitly check the binding between the end entity and the private key. Until operational protocols which do verify the binding (for both signature and encryption key pairs) exist, and are ubiquitous, this binding can only be assumed to be verified by the CA/RA. Therefore, if the binding is not verified by the CA/RA, certificates in the Internet Public-Key Infrastructure end up being somewhat less meaningful.

POP is accomplished in different ways depending on the type of key for which a certificate is requested. If a key can be used for multiple purposes (e.g. an RSA key) then any of the methods may be used.

This specification 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 have a signing key certified 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.

2.3.1 Signature Keys

For signature keys, the end entity can sign a value to prove possession of the private key.

2.3.2 Encryption Keys

For encryption keys, the end entity can provide the private key to the CA/RA, or can be required to decrypt a value in order to prove possession of the private key (see Section 3.2.8). Decrypting a value can be achieved either directly or indirectly.

The direct method is to issue a random challenge to which an immediate response is required.

The indirect method is to issue a certificate which is encrypted for the end entity (and have the end entity demonstrate its ability to decrypt this certificate in the confirmation message). This allows a CA to issue a certificate in a form which can only be used by the intended end entity.

This specification encourages the indirect method because this requires no extra messages to be sent (i.e., the proof can be demonstrated using the {request, response, confirmation} triple of messages).

2.3.3 Key Agreement Keys

For key agreement keys, the end entity and the PKI management entity (i.e. CA or RA) must establish a shared secret key in order to prove that the end entity has possession of the private key.

Note that this need not impose any restrictions on the keys which can be certified by a given CA -- in particular, for Diffie-Hellman keys the end entity may freely choose its algorithm parameters -- provided that the CA can generate a short-term (or one-time) key pair with the appropriate parameters when necessary.

2.4 Root CA key update

This discussion only applies to CAs which are a root CA for some end entity.

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 must generate two new cACertificate attribute values if certificates are made available using an X.500 directory.

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 who 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 entity's 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.

Notes:

1.This scheme does not make use of any of the X.509 v3 extensions as it must be able to work even for version 1 certificates. The presence of the KeyIdentifier extension would make for efficiency improvements.

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 entity's certificates, this generalization seems of dubious value. This 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.

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

2.4.1 CA Operator actions

To change the key of the CA, the CA operator 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).

3.Create a certificate containing the new CA public key signed with the old private key (the "new with old" certificate).

4.Create a certificate containing the new CA public key signed with the new private key (the "new with new" certificate).

5.Publish these new certificates via the directory and/or other means. (A CAKeyUpdAnn message.)

6.Export the new CA public key so that end entities may acquire it using the "out-of-band" mechanism.

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 is no longer required (other than for non-repudiation) will be when all end entities of this CA have acquired the new CA public key via "out-of-band" means.

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

The "new with old" certificate must 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.

The "new with new" certificate must have a validity period starting at

the generation time of the new key pair and ending at the time at which
the CA will next update its key pair.

2.4.2 Verifying Certificates.

Normally when verifying a signature the verifier verifies, among other things, the certificate containing the public key of the signer. However, once a CA is allowed to update its key there are a range of new possibilities. These are shown in the table below.

	Repository contains NEW and OLD public keys		Repository contains only OLD public key (due to e.g. delay in publication)	
	PSE Contains NEW public key	PSE Contains OLD public key	PSE Contains NEW public key	PSE Contains OLD public key
Signer's certificate is protected using NEW public key	Case 1: This is the standard case where the verifier can directly verify the certificate without using the directory	Case 3: In this case the verifier must access the directory in order to get the value of the NEW public key	Case 5: Although the CA operator has not updated the directory the verifier can verify the certificate directly - this is thus the same as case 1.	Case 7: In this case the CA operator has not updated the directory and so the verification will FAIL
Signer's certificate is protected using OLD public key	Case 2: In this case the verifier must access the directory in order to get the value of the OLD public key	Case 4: In this case the verifier can directly verify the certificate without using the directory	Case 6: The verifier thinks this is the situation of case 2 and will access the directory, however the verification will FAIL	Case 8: Although the CA operator has not updated the directory the verifier can verify the certificate directly - this is thus the same as case 4.

2.4.2.1 Verification in cases 1, 4, 5 and 8.

In these cases the verifier has a local copy of the CA public key which can be used to verify the certificate directly. This is the same as the situation where no key change has ever occurred.

Note that case 8 may arise between the time when the CA operator has generated the new key pair and the time when the CA operator stores the updated attributes in the directory. Case 5 can only arise if the CA operator has issued both the signer's and verifier's certificates during this "gap" (the CA operator should avoid this as it leads to the failure cases described below).

2.4.2.2 Verification in case 2.

In case 2 the verifier must get access to the old public key of the CA. The verifier does the following:

1.Lookup the CACertificate attribute in the directory and pick the appropriate value (based on validity periods)
2.Verify that this is correct using the new CA key (which the verifier has locally).
3.If correct then check the signer's certificate using the old CA key.

Case 2 will arise when the CA operator has issued the signer's certificate, then changed key and then issued the verifier's certificate, so it is quite a typical case.

2.4.2.3 Verification in case 3.

In case 3 the verifier must get access to the new public key of the CA. The verifier does the following:

1.Lookup the CACertificate attribute in the directory and pick the appropriate value (based on validity periods).
2.Verify that this is correct using the old CA key (which the verifier has stored locally).
3.If correct then check the signer's certificate using the new CA key.

Case 3 will arise when the CA operator has issued the verifier's certificate, then changed key and then issued the signer's certificate, so it is also quite a typical case.

2.4.2.4 Failure of verification in case 6.

In this case the CA has issued the verifier's PSE containing the new key without updating the directory attributes. This means that the verifier has no means to get a trustworthy version of the CA's old key and so verification fails.

Note that the failure is the CA operator's fault.

2.4.2.5 Failure of verification in case 7.

In this case the CA has issued the signer's certificate protected with the new key without updating the directory attributes. This means that the verifier has no means to get a trustworthy version of the CA's new key and so verification fails.

Note that the failure is again the CA operator's fault.

2.4.3 Revocation - Change of CA key

As we saw above the verification of a certificate becomes more complex once the CA is allowed to change its key. This is also true for revocation checks as the CA may have signed the CRL using a newer private key than the one that is within the user's PSE.

The analysis of the alternatives is as for certificate verification.

3. Data Structures

This section contains descriptions of the data structures required for PKI management messages. Section 4 describes constraints on their values and the sequence of events for each of the various PKI management operations. Section 5 describes how these may be encapsulated in various transport mechanisms.

3.1 Overall PKI Message

All of the messages used in PKI management use the following structure:

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

The PKIHeader contains information which is common to many PKI messages.

The PKIBody contains message-specific information.

The PKIProtection, when used, contains bits which protect the PKI message.

The extra certificates field can contain certificates which may be useful to the recipient. For example, this can be used by a CA or RA to present an end entity with certificates which it needs to verify it s own new certificate (if the CA that issued the end entity s certificate is not a root CA for the end entity).

Note also 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.

3.1.1 PKI Message Header

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

The following data structure is used to contain this information:

```
  PKIHeader ::= SEQUENCE {
      pvno                INTEGER     { ietf-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)
  }

  PKIFreeText ::= CHOICE {
      iA5String  [0] IA5String,
      bMPString  [1] BMPString
  } -- note that the text included here would ideally be in the
    -- preferred language of the recipient
```

The pvno field is fixed for this version of IPKI.

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., in the InitReqContent message, where the end entity may not know its own DN, e-mail name, IP address, etc.), then the "sender" field must contain a "NULL" value; that is, the SEQUENCE OF relative distinguished names is of zero length. In such a case the senderKID field must 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 usable to verify the protection on the message.

The protectionAlg field specifies the algorithm used to protect the message. If no protection bits are supplied (PKIProtection is optional) then this field must be omitted; if protection bits are supplied then this field must be supplied.

senderKID and recipKID are usable to indicate which keys have been used to protect the message (recipKID will normally only be required where protection of the message uses DH keys).

The transactionID field within the message header is required so that the recipient of a response message 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.

The messageTime field contains the time at which the sender created the message. This may be useful to allow end entities to correct their local time to be consistent with the time on a central system.

The freeText field may be used to send a human-readable message to the recipient.

3.1.2 PKI Message Body

```
  PKIBody ::= CHOICE {       -- message-specific body elements
      ir      [0]  InitReqContent,
      ip      [1]  InitRepContent,
      cr      [2]  CertReqContent,
      cp      [3]  CertRepContent,
      p10cr   [4]  PKCS10CertReqContent,
      popdecc [5]  POPODecKeyChallContent,
      popdecr [6]  POPODecKeyRespContent,
      kur     [7]  KeyUpdReqContent,
      kup     [8]  KeyUpdRepContent,
      krr     [9]  KeyRecReqContent,
      krp     [10] KeyRecRepContent,
      rr      [11] RevReqContent,
      rp      [12] RevRepContent,
      ccr     [13] CrossCertReqContent,
      ccp     [14] CrossCertRepContent,
      ckuann  [15] CAKeyUpdAnnContent,
      cann    [16] CertAnnContent,
      rann    [17] RevAnnContent,
      crlann  [18] CRLAnnContent,
      conf    [19] PKIConfirmContent,
      nested  [20] NestedMessageContent,
      infor   [21] PKIInfoReqContent,
      infop   [22] PKIInfoRepContent,
      error   [23] ErrorMsgContent
  }
```

The specific types are described in section 3.3 below.

3.1.3 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 the protectionBits 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 PKIX, will instead be applied. Such a choice is explicitly allowed in this specification. Examples of such external protection include PKCS #7 [PKCS7] and Security Multiparts [RFC1847] encapsulation of the PKIMessage. 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, key-recovery, or any other process with "boot-strapping" characteristics. For those cases it may be necessary that the PKIProtection parameter be used. In the future, if/when external mechanisms are modified to accommodate boot-strapping scenarios, the use of the PKIProtection parameter may become rare or non-existent.

Depending on the circumstances the PKIProtection bits may contain a MAC or signature. Only the following cases can occur:

- shared secret information

In this case the sender and recipient share secret information (established via out-of-band means or from a previous PKI management operation). The protection bits will typically contain a MAC value and the protectionAlg will be the following:

```
  PasswordBasedMac ::= OBJECT IDENTIFIER

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

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.]

- DH 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 PKI message. The protection bits will typically contain a MAC value keyed with this derived symmetric key and the protectionAlg will be the following:.

```
  DHBasedMac ::= OBJECT IDENTIFIER

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

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.]

- signature

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

- multiple protection

In cases where an end entity sends a protected PKI message to an RA, the RA may forward that message to a CA, attaching it s own protection. This is accomplished by nesting the entire message sent by the end entity within a new PKI message. The structure used is as follows.

```
  NestedMessageContent ::= ANY
  -- This will be a PKIMessage
```

3.2 Common Data Structures

Before specifying the specific types which may be placed in a PKIBody we define some useful data structures which are used in more than one case.

3.2.1 Requested Certificate Contents

Various PKI management messages require that the originator of the message indicate some of the fields which are required to be present in a certificate. The CertTemplate structure allows an end entity or RA to specify as much as they wish about the certificate it requires. ReqCertContent is basically the same as 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.

```
  CertTemplate ::= SEQUENCE {
      version    [0] Version               OPTIONAL,
      -- used to ask for a particular syntax version
      serial     [1] INTEGER               OPTIONAL,
      -- used to ask for a particular serial number
      signingAlg [2] AlgorithmIdentifier   OPTIONAL,
      -- used to ask the CA to use this alg. for signing the cert
      subject    [3] Name                  OPTIONAL,
      validity   [4] OptionalValidity      OPTIONAL,
      issuer     [5] Name                  OPTIONAL,
      publicKey  [6] SubjectPublicKeyInfo  OPTIONAL,
      issuerUID  [7] UniqueIdentifier      OPTIONAL,
      subjectUID [8] UniqueIdentifier      OPTIONAL,
      extensions [9] Extensions            OPTIONAL
      -- the extensions which the requester would like in the cert.
  }

  OptionalValidity ::= SEQUENCE {
      notBefore  [0] UTCTime OPTIONAL,
      notAfter   [1] UTCTime OPTIONAL
  }
```

3.2.2 Encrypted Values

Where encrypted values (restricted, in this specification, to be either private keys or certificates) are sent in PKI messages the following data structure is used.

```
  EncryptedValue ::= SEQUENCE {
      encValue          BIT STRING,
      -- the encrypted value itself
      intendedAlg   [0] AlgorithmIdentifier  OPTIONAL,
      -- the intended algorithm for which the value will be used
      symmAlg       [1] AlgorithmIdentifier  OPTIONAL,
      -- the symmetric algorithm used to encrypt the value
      encSymmKey    [2] BIT STRING           OPTIONAL,
      -- the (encrypted) symmetric key used to encrypt the value
      keyAlg        [3] AlgorithmIdentifier  OPTIONAL
      -- algorithm used to encrypt the symmetric key
  }
```

Use of this data structure requires that the creator and intended recipient are respectively 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.

3.2.3 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
      -- FullCertTemplate
  }
```

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

```
  PKIFailureInfo ::= BIT STRING {
  -- since we 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
  }
```

3.2.4 Certificate Identification

In order to identify particular certificates the following data structure is used.

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

3.2.5 "Out-of-band" root CA public key

Each root 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 document, we define data structures which can support such mechanisms.

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 of this value to allow verification of its integrity before use.

```
OOBCert ::= Certificate
```

The fields within this certificate are restricted as follows:

- The certificate must be self-signed, i.e. the signature must be verifiable using the subjectPublicKey field.
- The subject and issuer fields should be identical.
- If the subject field is NULL then both subjectAltNames and issuerAltNames extensions must be present and have exactly the same value.
- The values of all other extensions must be suitable for a self-certificate (e.g. key identifiers for subject and issuer must 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 value here is that anyone who has securely gotten the hash value (via the out-of-band means) can verify a self-signed certificate for that CA.

3.2.6 Archival Options

Requesters may indicate that they wish the PKI to archive a private key value using the following structure:

```
PKIArchiveOptions ::= CHOICE {
    encryptedPrivKey     [0] EncryptedValue,
    -- the actual value of the private key
    keyGenParameters     [1] KeyGenParameters,
    -- parameters which allow the private key to be re-generated
    archiveRemGenPrivKey [2] BOOLEAN
    -- set to TRUE if sender wishes receiver to archive the private
    -- key of a key pair which the receiver generates in response to
    -- this request; set to FALSE if no archival is desired.
}
```

```
  KeyGenParameters ::= OCTET STRING
      -- an alternative to sending the key is to send the information
      -- about how to re-generate the key (e.g. for many RSA
      -- implementations one could send the first random numbers tested
      -- for primality).
      -- The actual syntax for this parameter may be defined in a
      -- subsequent version of this document or in another standard.
```

3.2.7 **Publication Information**

Requesters may indicate that they wish the PKI to publish a certificate using the structure below.

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, the requester indicates that the PKI may publish the certificate using whatever means it chooses.

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).

```
  PKIPublicationInfo ::= SEQUENCE {
     action     INTEGER {
                  dontPublish (0),
                  pleasePublish (1)
                },
     pubInfos  SEQUENCE OF SinglePubInfo OPTIONAL
       -- pubInfos must 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)
      },
      pubLocation  GeneralName OPTIONAL
  }
```

3.2.8 **"Full" Request Template**

The following structure groups together the fields which may be sent as part of a certification request:

```
FullCertTemplates ::= SEQUENCE OF FullCertTemplate
```

```
FullCertTemplate ::= SEQUENCE {
    certReqId              INTEGER,
    -- to match this request with corresponding response
    -- (note:  must be unique over all FullCertReqs in this message)
    certTemplate           CertTemplate,
    popoPrivKeyVerified    BOOLEAN DEFAULT FALSE,
    popoSigningKey     [0] POPOSigningKey      OPTIONAL,
    archiveOptions     [1] PKIArchiveOptions   OPTIONAL,
    publicationInfo    [2] PKIPublicationInfo  OPTIONAL,
    oldCertId          [3] CertId              OPTIONAL
    -- id. of cert. which is being updated by this one
}
```

When the certification request is made by an RA on behalf of some other end entity, then the RA may indicate to the CA that it has already verified proof-of-possession (of the private key corresponding to the public key for which a certificate is being requested) by setting popoPrivKeyVerified to TRUE. If the proof-of-possession has not yet been verified, or if the request is not being made by an RA, then the popoPrivKeyVerified field is omitted (defaulting to FALSE) and the popoSigningKey field or the challenge-response protocol described below may be used to prove possession (depending on the type of key involved).

If the certification request is for a signing key pair (i.e., a request for a verification certificate), then the proof of possession of the private signing key is demonstrated through use of the POPOSigningKey structure.

```
POPOSigningKey ::= SEQUENCE {
    poposkInput         POPOSKInput,
    alg                 AlgorithmIdentifier,
    signature           BIT STRING
    -- the signature (using "alg") on the DER-encoded
    -- value of poposkInput
}

POPOSKInput ::= CHOICE {
    popoSigningKeyInput      [0] POPOSigningKeyInput,
    certificationRequestInfo     CertificationRequestInfo
    -- imported from [PKCS10] (note that if this choice is used,
    -- POPOSigningKey is simply a standard PKCS #10 request; this
    -- allows a bare PKCS #10 request to be augmented with other
    -- desired information in the FullCertTemplate before being
    -- sent to the CA/RA)
}
```

```
  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] BIT STRING
          -- used if no authenticated GeneralName currently exists for
          -- the sender; publicKeyMAC contains a password-based MAC
          -- (using the protectionAlg AlgId from PKIHeader) on the
          -- DER-encoded value of publicKey
      },
      publicKey           SubjectPublicKeyInfo    -- from CertTemplate
  }
```

On the other hand, if the certification request is for an encryption key pair (i.e., a request for an encryption certificate), then the proof of possession of the private decryption key may be demonstrated in one of three ways.

1) By the inclusion of the private key (encrypted) in the FullCertTemplate (in the PKIArchivalOptions structure).

2) By having the CA return not the certificate, but an encrypted certificate (i.e., the certificate encrypted under a randomly-generated symmetric key, and the symmetric key encrypted under the public key for which the certification request is being made). The end entity proves knowledge of the private decryption key to the CA by MACing the PKIConfirm message using a key derived from this symmetric key. [Note that if several FullCertTemplates are included in the PKIMessage, then the CA uses a different symmetric key for each FullCertTemplate and the MAC uses a key derived from the concatenation of all these keys.] The MACing procedure uses the PasswordBasedMac AlgId defined in Section 3.1.

3) By having the end entity engage in a challenge-response protocol (using the messages POPODecKeyChallContent and POPODecKeyRespContent) between the CertReq and CertRep messages. [This method would typically be used in an environment in which an RA verifies POP and then makes a certification request to the CA on behalf of the end entity. In such a scenario, the CA trusts the RA to have done POP correctly before the RA requests a certificate for the end entity.] The complete protocol then looks as follows (note that req' does not necessarily encapsulate req as a nested message):

```
                     EE             RA             CA
                      ---- req ---->
                      <--- chall ---
                      ---- resp --->
                                    ---- req' --->
                                    <--- rep -----
                                    ---- conf --->
                      <--- rep -----
                      ---- conf --->
```

This protocol is obviously much longer than the 3-way exchange given in choice (2) above, but allows a local Registration Authority to be involved and has the property that the certificate itself is not actually created until the proof of possession is complete.

3.3 Operation-Specific Data Structures

3.3.1 Initialization Request

An Initialization request message (InitReq) contains an InitReqContent 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 (see Appendix B profiles for further information).

```
  InitReqContent ::= SEQUENCE {
      protocolEncKey      [0] SubjectPublicKeyInfo  OPTIONAL,
      fullCertTemplates       FullCertTemplates
  }
```

3.3.2 Initialization Response

An Initialization response message (InitRep) contains an InitRepContent 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).

```
  InitRepContent ::= CertRepContent
```

3.3.3 Registration/Certification Request

A Registration/Certification request message (CertReq) contains a CertReqContent data structure which specifies the requested FullCertTemplates.

Alternatively, for the cases in which it can be used, the CertReq may contain a PKCS10CertReqContent. This structure is fully specified by the ASN.1 structure CertificationRequest given in [PKCS10].

```
  CertReqContent ::= CHOICE {
      fullCertTemplates    [0] FullCertTemplates,
      pkcs10CertReqContent [1] PKCS10CertReqContent
  }
```

The challenge-response messages for proof of possession of a private decryption key are specified as follows (see [MvOV97, p.404], for details). Note that this challenge-response exchange is associated with the preceding cert. request message (and subsequent cert. response and confirmation messages) by the nonces used in the PKIHeader and by the protection (MACing or signing) applied to the PKIMessage.

```
  POPODecKeyChallContent ::= SEQUENCE OF Challenge
  -- One Challenge per encryption key certification request (in the
  -- same order as these requests appear in FullCertTemplates).

  Challenge ::= SEQUENCE {
      owf                 AlgorithmIdentifier  OPTIONAL,
      -- must be present in the first Challenge; may be omitted in any
      -- subsequent Challenge in POPODecKeyChallContent (if omitted,
      -- then the owf used in the immediately preceding Challenge is
      -- to be used).
      witness             OCTET STRING,
      -- the result of applying the one-way function (owf) to a
      -- randomly-generated INTEGER, A.  [Note that a different
      -- INTEGER must be used for each Challenge.]
      challenge           OCTET STRING
      -- the encryption (under the public key for which the cert.
      -- request is being made) of Rand, where Rand is specified as
      --   Rand ::= SEQUENCE {
      --      int      INTEGER,
      --       - the randomly-generated INTEGER A (above)
      --      sender   GeneralName
      --       - the sender's name (as included in PKIHeader)
      --   }
  }

  POPODecKeyRespContent ::= SEQUENCE OF INTEGER
  -- One INTEGER per encryption key certification request (in the
  -- same order as these requests appear in FullCertTemplates).  The
  -- retrieved INTEGER A (above) is returned to the sender of the
  -- corresponding Challenge.
```

3.3.4 Registration/Certification Response

A registration response message (CertRep) contains a CertRepContent data structure which has a CA public key, a status value and optionally failure information, a subject certificate, and an encrypted private key.

```
  CertRepContent ::= SEQUENCE {
      caPub           [1] Certificate             OPTIONAL,
      response            SEQUENCE OF CertResponse
  }

  CertResponse ::= SEQUENCE {
      certReqId           INTEGER,
      -- to match this response with corresponding request
      status              PKIStatusInfo,
      certifiedKeyPair    CertifiedKeyPair    OPTIONAL
  }

  CertifiedKeyPair ::= SEQUENCE {
      certificate     [0] Certificate         OPTIONAL,
      encryptedCert   [1] EncryptedValue      OPTIONAL,
      privateKey      [2] EncryptedValue      OPTIONAL,
      publicationInfo [3] PKIPublicationInfo  OPTIONAL
  }
```

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.

The CertifiedKeyPair structure must contain either a Certificate or an EncryptedCert, and an optional EncryptedPrivateKey (i.e. not both a Certificate and EncryptedCert).

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.

3.3.5 Key update request content

For key update requests the following syntax is used. Typically, SubjectPublicKeyInfo, KeyId, and Validity are the template fields which may be supplied for each key to be updated.

```
  KeyUpdReqContent ::= SEQUENCE {
      protocolEncKey      [0] SubjectPublicKeyInfo  OPTIONAL,
      fullCertTemplates   [1] FullCertTemplates     OPTIONAL
```

```
   }
```

3.3.6 Key Update response content

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

```
  KeyUpdRepContent ::= InitRepContent
```

3.3.7 Key Recovery Request content

For key recovery requests the syntax used is identical to the initialization request InitReqContent. Typically, SubjectPublicKeyInfo and KeyId are the template fields which may be used to supply a signature public key for which a certificate is required (see Appendix B profiles for further information).

```
  KeyRecReqContent ::= InitReqContent
```

3.3.8 Key recovery response content

For key recovery responses the following syntax is used. For some status values (e.g., waiting) none of the optional fields will be present.

```
  KeyRecRepContent ::= SEQUENCE {
      status                  PKIStatusInfo,
      newSigCert          [0] Certificate                   OPTIONAL,
      caCerts             [1] SEQUENCE OF Certificate       OPTIONAL,
      keyPairHist         [2] SEQUENCE OF CertifiedKeyPair  OPTIONAL
  }
```

3.3.9 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,
      -- from the DAM, so that CA knows which Dist. point to use
      badSinceDate        GeneralizedTime  OPTIONAL,
      -- indicates best knowledge of sender
      crlEntryDetails     Extensions
      -- requested crlEntryExtensions
```

```
   }
```

3.3.10 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              PKIStatusInfo,
      revCerts        [0] SEQUENCE OF CertId OPTIONAL,
      -- identifies the certs for which revocation was requested
      crls            [1] SEQUENCE OF CertificateList  OPTIONAL
      -- the resulting CRLs (there may be more than one)
  }
```

3.3.11 Cross certification request content

Cross certification requests use the same syntax as for normal certification requests with the restriction that the key pair must have been generated by the requesting CA and the private key must not be sent to the responding CA.

```
  CrossCertReqContent ::= CertReqContent
```

3.3.12 Cross certification response content

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

```
  CrossCertRepContent ::= CertRepContent
```

3.3.13 CA 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
  }
```

3.3.14 Certificate Announcement

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

Note that this structure (and the CertAnn message itself) 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
```

3.3.15 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.

3.3.16 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
```

3.3.17 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
```

3.3.18 PKI Information Request content

```
  InfoTypeAndValue ::= SEQUENCE {
      infoType               OBJECT IDENTIFIER,
      infoValue              ANY DEFINED BY infoType  OPTIONAL
  }
  -- Example InfoTypeAndValue contents include, but are not limited to:
  --    { CAProtEncCert    = { xx }, Certificate                     }
  --    { SignKeyPairTypes = { xx }, SEQUENCE OF AlgorithmIdentifier }
  --    { EncKeyPairTypes  = { xx }, SEQUENCE OF AlgorithmIdentifier }
```

```
--   { PreferredSymmAlg = { xx }, AlgorithmIdentifier             }
--   { CAKeyUpdateInfo  = { xx }, CAKeyUpdAnnContent              }
--   { CurrentCRL       = { xx }, CertificateList                 }
```

```
  PKIInfoReqContent ::= SET OF InfoTypeAndValue
  -- The OPTIONAL infoValue parameter of InfoTypeAndValue is unused.
  -- The CA is free to ignore any contained OBJ. IDs that it does not
  -- recognize.
  -- The empty set indicates that the CA may send any/all information
  -- that it wishes.
```

3.3.19 PKI Information Response content

```
  PKIInfoRepContent ::= SET OF InfoTypeAndValue
  -- The end entity is free to ignore any contained OBJ. IDs that it
  -- does not recognize.
```

3.3.20 Error Message content

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

4. PKI Management functions

The PKI management functions outlined in section 1 above are described in this section.

This section is split into two, the first part dealing with functions which are "mandatory" in the sense that all end entity and CA/RA implementations must be able to provide functionality described via one of the transport mechanisms defined in section 5. This part is effectively the profile of the PKI management functionality which must be supported.

The second part defines "additional" functions.

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

4.1 Mandatory Functions

4.1.1 Root CA initialisation

A newly created root CA must produce a "self-certificate" which is a Certificate structure with the profile defined for the "newWithNew" certificate issued following a root CA key update.

In order to make the CA s self certificate useful to end entities which do not acquire this information via "out-of-band" means, the CA must also produce a fingerprint for its public key. End entities which acquire this value securely via some "out-of-band" means can then verify the CA s self-certificate and hence the other attributes contained therein.

The data structure used to carry the fingerprint is the OOBCertHash.

The root CA must also produce an initial revocation list.

4.1.2 Root CA key update

4.1.3 Subordinate CA initialisation

From the perspective of PKI management protocols the initialisation of a subordinate CA is the same as the initialisation of an end entity. The only difference is that the subordinate CA must also produce an initial revocation list.

4.1.4 CRL production

Before issuing any certificates a newly established CA (which issues CRLs) must produce "empty" versions of each CRL which is to be periodically produced.

4.1.5 PKI information request

The above operations produce various data structures which are used in PKI management protocols.

When a PKI entity wishes to acquire information about the current status of a CA it may send that CA a PKIInfoReq PKIMessage. The response will be a PKIInfoRep message.

The CA must respond to the request with a response providing (at least) all of the information requested by the requester. If some of the information cannot be provided then an error message must be returned.

The PKIInfoReq and PKIInfoRep 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).

4.1.6 Cross certification

The initiating CA is the CA which will become the subject of the cross-certificate, the responding CA will become the issuer of the cross-certificate.

The initiating CA must be "up and running" before initiating the cross-certification operation.

As with registration/certification there are a few possibilities here.

4.1.6.1 One-way request-response scheme:

The cross-certification scheme is essentially a one way operation; that is, when successful, this operation 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 operation (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.

The followings steps occur:

1. The initiating CA gathers the information required for the cross certification request;
2. The initiating CA creates the cross-certification request message (CrossCertReq);
3. The CrossCertReq message is transported to the responding CA;
4. The responding CA processes the CrossCertReq -- this results in the creation of a cross-certification response (CrossCertRep) message;
5. The CrossCertRep message is transported to the initiating CA;
6. The initiating CA processes the CrossCertRep (depending on its content some looping may be required; that is, the initiating CA may have to await further responses or generate a new CrossCertReq for the responding CA);
7. The initiating CA creates a PKIConfirm message and transports it to the responding CA.

Notes:

1. The CrossCertReq must contain a "complete" certification request, that is, all fields (including e.g. a BasicConstraints extension) must be specified by the initiating CA.
2. The CrossCertRep message should contain the verification certificate of the responding CA - if present, the initiating CA must then verify this certificate (for example, via the "out-of-band" mechanism).

4.1.7 End entity initialisation

As with CAs, end entity s must be initialised. Initialisation of end entities requires at least two steps:

- acquisition of PKI information
- out-of-band verification of one root-CA public key

(other possible steps include the retrieval of trust condition information and/or out-of-band verification of other CA public keys).

4.1.7.1 Acquisition of PKI information

The information required is:

- the current root-CA public key
- (if the certifying CA is not a root-CA) the certification path from the root CA to the certifying CA together with appropriate revocation lists
- the algorithms and algorithm parameters which the certifying CA supports for each relevant usage

Additional information could be required (e.g. supported extensions or CA policy information) in order to produce a certification request which will be successful. However, for simplicity we do not mandate that the end entity acquires this information via the PKI messages. The end result is simply that some certification requests may fail (e.g., if the end entity wants to generate its own encryption key but the CA doesn t allow that).

The required information is acquired as follows (see Section 3.3.18):

- the end entity sends a pKIInfoReq to the certifying CA requesting the information it requires;

- the certifying CA responds with a pKIInfoRep message which contains the requested information.

4.1.8 Certificate Update

When a certificate is due to expire the relevant end entity may request that the CA update the certificate - that is, that the CA issue a new certificate which differs from the previous one only in terms of PKI attributes (serialNumber, validity, some extensions) and is otherwise identical.

Two options must be catered for here, where the end entity initiates this operation, and where the CA initiates the operation and then creates a message informing the end entity of the existence of the new certificate.

4.2 Additional Functions

4.2.1 Cross certification

4.2.1.1 Two-way request-response scheme:

4.2.1.1.1 Overview of Exchange

This cross certification exchange allows two CAs to simultaneously certify each other. This means that each CA will create a certificate that contains the CA verification key of the other CA.

Cross certification is initiated at one CA known as the responder. The CA administrator for the responder identifies the CA it wants to cross certify and the responder CA equipment generates an authorization code. The responder CA administrator passes this authorization code by out-of-band means to the requester CA administrator. The requester CA administrator enters the authorization code at the requester CA 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.

Serial numbers and protocol version are used in the same manner as in the above CA-client exchanges.

4.2.1.1.2 Detailed Description

The requester CA initiates the exchange by generating a random number (requester random number). The requester CA then sends the responder CA the message CrossReq. The fields in this message are protected from modification with a MAC based on the authorization code.

Upon receipt of the CrossReq message, the responder CA 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 a new requester certificate which contains the requester CA public key and is signed with the responder CA signature private key. The responder CA responds with the message CrossRep. The fields in this message are protected from modification with a MAC based on the authorization code.

Upon receipt of the CrossRep message, the requester CA checks that its own system time is close to the responder CA system time, checks the received random numbers and validates the MAC. It then generates and archives a new responder certificate which contains the responder CA public key and is signed by the requester CA signature private key. The requester CA responds with the message PKIConfirm. The fields in this message are protected from modification with a MAC based on the authorization code.

Upon receipt of the PKIConfirm message, the responder CA checks the random numbers, archives the responder certificate, and validates the MAC. It writes both the request and responder certificates to the Directory. It then responds with its own PKIConfirm message. The fields in this message are protected from modification with a MAC based on the authorization code.

Upon receipt of the PKIConfirm message, the requester CA checks the

random numbers and validates the MAC. The requester CA writes both the
requester and responder certificates to the Directory.

4.2.2 End entity initialisation

As with CAs, end entities must be initialised. Initialisation of end entities requires two steps:

- acquisition of PKI information
- out-of-band verification of root-CA public key

4.2.2.1 Acquisition of PKI information

See previous section.

4.2.2.2 Import of CA key fingerprint

An end entity must securely possess the public key of its root CA. One method to achieve this is to provide the end entity with the CA s self-certificate fingerprint via some secure "out-of-band" means. The end entity can then securely use the CA s self-certificate.

The data structure used is the OOBcertHash

5. Transports

The transport protocols specified below allow 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).

5.1 File based protocol

A file containing a PKI message must contain only the DER encoding of one PKI message, i.e. there must be no extraneous header or trailer information in the file.

Such files can be used to transport PKI messages using e.g. FTP.

5.2 Socket based Management Protocol

The following simple socket based protocol is to be used for transport of PKI messages. This protocol is suitable for cases where an end entity (or an RA) initiates a transaction and can poll to pick up the results.

If a transaction is initiated by a PKI entity (RA or CA) then an end entity must either supply a listener process or be supplied with a polling reference (see below) in order to allow it to pick up the PKI message from the PKI management component.

The protocol basically assumes a listener process on an RA or CA which can accept PKI messages on a well-defined port (port number 829). Typically an initiator binds to this port and submits the initial PKI message for a given transaction ID. The responder replies with a PKI message and/or with a reference number to be used later when polling for the actual PKI message response.

If a number of PKI response messages are to be produced for a given request (say if some part of the request is handled more quickly than another) then a new polling reference is also returned.

When the final PKI response message has been picked up by the initiator then no new polling reference is supplied.

The initiator of a transaction sends a "socket PKI message" to the recipient. The recipient responds with a similar message.

A "socket PKI message" consists of:

      length (32-bits), flag (8-bits), value (defined below)

The length field contains the number of octets of the remainder of the message (i.e., number of octets of "value" plus one).

```
 Message name   flag     value

 msgReq          00 H    DER-encoded PKI message
   -- PKI message from initiator
 pollRep         01 H    polling reference (32 bits),
                         time-to-check-back (32 bits)
   -- poll response where no PKI message response ready; use polling
   -- reference value (and estimated time value) for later polling
 pollReq         02 H    polling reference (32 bits)
   -- request for a PKI message response to initial message
 negPollRep      03 H     00 H
   -- no further polling responses (i.e., transaction complete)
 partialMsgRep   04 H    next polling reference (32 bits),
                         time-to-check-back (32 bits),
                         DER-encoded PKI message
   -- partial response to initial message plus new polling reference
   -- (and estimated time value) to use to get next part of response
 finalMsgRep     05 H    DER-encoded PKI message
   -- final (and possibly sole) response to initial message
 errorMsgRep     06 H    human readable error message
   -- produced when an error is detected (e.g., a polling reference is
   -- received which doesn t exist or is finished with)
```

Where a PKIConfirm message is to be transported (always from the initiator to the responder) then a msgReq message is sent and a negPollRep is returned.

The sequence of messages which can occur is then:

a) end entity sends msgReq and receives one of pollRep, negPollRep, partialMsgRep or finalMsgRep in response.
b) end entity sends pollReq message and receives one of negPollRep, partialMsgRep, finalMsgRep or ErrorMsgRep in response.

The "time-to-check-back" parameter is a 32-bit integer, defined to be the number of seconds which have elapsed since midnight, January 1, 1970, coordinated universal time. It provides an estimate of the time that the end entity should send its next pollReq.

5.3 Management Protocol via E-mail

This subsection specifies a means for conveying ASN.1-encoded messages for the protocol exchanges described in Section 4 via Internet mail.

A simple MIME object is specified as follows.

```
Content-Type: application/x-pkix3
Content-Transfer-Encoding: base64

<<the ASN.1 DER-encoded PKIX-3 message, base64-encoded>>
```

This MIME object can be sent and received using common MIME processing engines and provides a simple Internet mail transport for PKIX-3 messages.

5.4 Management Protocol via HTTP

This subsection specifies a means for conveying ASN.1-encoded messages for the protocol exchanges described in Section 4 via the HyperText Transfer Protocol.

A simple MIME object is specified as follows.

```
Content-Type: application/x-pkix3

<<the ASN.1 DER-encoded PKIX-3 message>>
```

This MIME object can be sent and received using common HTTP processing engines over WWW links and provides a simple browser-server transport for PKIX-3 messages.

SECURITY CONSIDERATIONS

This entire memo is about security mechanisms.

One cryptographic consideration is worth explicitly spelling out. In the protocols specified above, when an end entity is required to prove possession of a decryption key, it is effectively challenged to decrypt something (its own certificate). This scheme (and many others!) could be vulnerable to an attack if the possessor of the decryption key in question could be fooled into decrypting an arbitrary challenge and returning the cleartext to an attacker. Although in this specification a number of other failures in security are required in order for this attack to succeed, it is conceivable that some future services (e.g., notary, trusted time) could potentially be vulnerable to such attacks. For this reason we re-iterate the general rule that implementations should be very careful about decrypting arbitrary "ciphertext" and revealing recovered "plaintext" since such a practice can lead to serious security vulnerabilities.

References

[MvOV97] A. Menezes, P. van Oorschot, S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1997.

[PKCS7] RSA Laboratories, "The Public-Key Cryptography Standards (PKCS)", RSA Data Security Inc., Redwood City, California, November 1993 Release.

[PKCS10] RSA Laboratories, "The Public-Key Cryptography Standards (PKCS)", RSA Data Security Inc., Redwood City, California, November 1993 Release.

[PKCS11] RSA Laboratories, "The Public-Key Cryptography Standards - PKCS #11: Cryptographic token interface standard", RSA Data Security Inc., Redwood City, California, April 28, 1995.

[PKIX-2] S. Boeyen, R. Housley, T. Howes, M. Myers, P. Richard, "Internet Public Key Infrastructure Part 2: Operational Protocols", Internet Draft draft-ietf-pkix-ipki2opp-0x.txt (work in progress).

[RFC1847] J. Galvin, S. Murphy, S. Crocker, N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/

Encrypted", Internet Request for Comments 1847, October
1995.

Authors' Addresses

Carlisle Adams
Entrust Technologies
750 Heron Road
Ottawa, Ontario
Canada K1V 1A7
cadams@entrust.com

Stephen Farrell
Software and Systems Engineering Ltd.
Fitzwilliam Court
Leeson Close
Dublin 2
IRELAND
stephen.farrell@sse.ie

APPENDIX A: Reasons for the presence of RAs

The reasons which justify the presence of an RA can be split into those which are due to technical factors and those which are organizational in nature. Technical reasons include the following.

- If hardware tokens are in use, then not all end entities will have the equipment needed to initialize these; the RA equipment can include the necessary functionality (this may also be a matter of policy).

- Some end entities may not have the capability to publish certificates; again, the RA may be suitably placed for this.

- The RA will be able to issue signed revocation requests on behalf of end entities associated with it, whereas the end entity may not be able to do this (if the key pair is completely lost).

Some of the organisational reasons which argue for the presence of an RA are the following.

- It may be more cost effective to concentrate functionality in the RA equipment than to supply functionality to all end entities (especially if special token initialization equipment is to be used).

- Establishing RAs within an organization can reduce the number of CAs required, which is sometimes desirable.

- RAs may be better placed to identify people with their "electronic" names, especially if the CA is physically remote from the end entity.

- For many applications there will already be in place some administrative structure so that candidates for the role of RA are easy to find (which may not be true of the CA).

Appendix B. PKI management message profiles.

This appendix contains detailed profiles for those PKIMessages which must be supported by conforming implementations.

Profiles for the PKIMessages used in the following PKI management operations are provided:

- root CA key update
- information request/reponse
- cross-certification (1-way)
- initial registration and certification
    - centralised scheme
    - basic authenticated scheme

<<Later revisions will extend the above to include profiles for the operations listed below>>

- certificate update
    - end entity initiated
    - PKI initiated
- key update
- revocation request
- certificate publication
- CRL publication

B1. General Rules for interpretation of these profiles.

1. **Where OPTIONAL or DEFAULT fields are not mentioned in individual profiles, they should be absent from the relevant message.** Mandatory fields are not mentioned if they have an obvious value (e.g., pvno).
2. **Where structures occur in more than one message, they are separately profiled as appropriate.**
3. **The algorithmIdentifiers from PKIMessage structures are profiled separately.**
4. **A "special" X.500 DN is called the "NULL-DN"; this means a DN containing a zero-length SEQUENCE OF rdns (it s DER encoding is** then 3000 H).
5. **Where a GeneralName is required for a field but no suitable value is available (e.g. an end entity produces a request before** knowing its name) then the GeneralName is to be an X.500 NULL-DN (i.e., the Name field of the CHOICE is to contain a NULL-DN). This special value can be called a "NULL-GeneralName".
6. **Where a profile omits to specify the value for a GeneralName then the NULL-GeneralName value is to be present in the relevant** PKIMessage field. This occurs with the sender field of the PKIHeader for some messages.
7. **Where any ambiguity arises due to naming of fields, the profile**

names these using a "dot" notation (e.g., "certTemplate.subject" means the subject field within a field called certTemplate).

8. **Where a "SEQUENCE OF types" is part of a message, a zero-based array notation is used to describe fields within the SEQUENCE OF** (e.g., FullCertTemplates[0].certTemplate.subject refers to a subfield of the first FullCertTemplate contained in a request message).
9. **All PKI message exchanges (other than the centralised initial registration/certification scheme) require a PKIConfirm message** to be sent by the initiating entity. This message is not included in many of the profiles given below since its body is NULL and its header contents are clear from the context. Any authenticated means can be used for the protectionAlg (e.g., password-based MAC, if shared secret information is known, or signature).

B2. Algorithm Use Profile

The following table contains definitions of algorithm uses within PKI management protocols.

The columns in the table are:

Name: an identifier used for message profiles
Use: description of where and for what the algorithm is used
Mandatory: an AlgorithmIdentifier which must be supported by conforming implementations
Others: alternatives to the mandatory AlgorithmIdentifier

Name	Use	Mandatory	Others
CA_FP_ALG	Calculation of root CA public key fingerprint	SHA-1 + ASCII mapping	MD5,...
MSG_SIG_ALG	Protection of PKI messages using signature	RSA/SHA-1	RSA/MD5...
MSG_MAC_ALG	protection of PKI messages using MACing	HMAC	X9.9...
SYM_PENC_ALG	symmetric encryption of an end entity s private key where symmetric key is distributed out-of-band	3-DES (3-key-EDE, CBC mode)	RC5,CAST...
PROT_ENC_ALG	asymmetric algorithm used for encryption of (symmetric keys for encryption of) private keys transported in PKIMessages	RSA	D-H
PROT_SYM_ALG	symmetric encryption algorithm used for	3-DES (3-key-EDE, CBC mode)	RC5,CAST...

encryption of private
key bits (a key of this
type is encrypted using
PROT_ENC_ALG)

B3. "Self-signed" certificates

Profile of how a Certificate structure may be "self-signed". These strucures are used for distribution of "root" CA public keys. This can occur in one of three ways (see section 2.4 above for a description of the use of these structures):

Type	Function
newWithNew	a true "self-signed" certificate; the contained public key must be usable to verify the signature (though this provides only integrity and no authentication whatsoever)
oldWithNew	previous root CA public key signed with new private key
newWithOld	new root CA public key signed with previous private key

<<profile of certificate in such cases including relevant extensions, e.g. when present subjectAltName must be identical to issuerAltName, keyIdentifiers if present must contain appropriate values, etc.>>

B4. Proof of Possession Profile

"popo" fields for use when proving possession of a private signing key which corresponds to a public verification key for which a certificate has been requested.

Field	Value	Comment
alg	MSG_SIG_ALG	only signature protection is allowed for this proof
signature	present	bits calculated using MSG_SIG_ALG

<<Proof of possession of a private decryption key which corresponds to a public encryption key for which a certificate has been requested does not use this profile; instead the method given in protectionAlg for PKIConfirm in Section B.8.2 is used.>>

Not every CA/RA will require Proof-of-Possession (of signing key or of decryption key) in the certification request protocol. Although this specification STRONGLY RECOMMENDS that POP be verified by the CA/RA (because created certificates become less meaningful in the PKI otherwise; see Section 2.3), this may ultimately be a policy issue which is made explicit for any given CA in its publicized Policy OID and Certification Practice Statement. All end entities must be prepared to provide POP (i.e., these components of the PKIX-3 protocol must be supported).

CAs/RAs may therefore conceptually be divided into two classes (those which require POP as a condition of certificate creation and those which do not). End entities may choose to make verification decisions (as one step in certificate chain processing) at least partly by considering which classes of CAs (as indicated, for example, by their policy OIDs or Certification Practice Statements) have created the certificates included in the chain.

B5. Root CA Key upate

A root CA updates its key pair. It then produces a CA key update announcement message which can be made available (via one of the transport mechanisms) to the relevant end entities.

ckuann message:

Field	Value	Comment
sender	CA name	responding CA name
body	ckuann(CAKeyUpdAnnContent)	
oldWithNew	present	see section B.0 above
newWithOld	present	see section B.0 above
newWithNew	present	see section B.0 above
extraCerts	optionally present	can be used to "publish" certificates (e.g., certificates signed using the new private key)

B6. PKI Information request/response

End entity sends information request to PKI requesting details which will be required for later PKI managment operations. RA/CA responds with information response. If an RA generates the response then it will simply forward the equivalent message which it previously received from the CA, with the possible addition of the certificates to the extracerts fields of the PKIMessage.

Message Flows:

Step#	End entity				PKI
1	**format infor**				
2		**->**	infor	->	
3					**handle infor**
4					**produce infop**
5		**<-**	infop	<-	
6	**handle infop**				

```
infor:

Field               Value

recipient           CA name
  -- the name of the CA as contained in issuerAltName extensions or
  -- issuer fields within certificates
protectionAlg       MSG_MAC_ALG or MSG_SIG_ALG
  -- any authenticated protection alg.
SenderKID           present if required
  -- must be present if required for verification of message protection
freeText            any valid value
body                infor (PKIInfoReqContent)
PKIInfoReqContent   empty SET
  -- all relevant information requested
protection          present
  -- bits calculated using MSG_MAC_ALG or MSG_SIG_ALG



infop:

Field                Value

sender               CA name
  -- name of the CA which produced the message
protectionAlg        MSG_MAC_ALG or MSG_SIG_ALG
  -- any authenticated protection alg.
senderKID            present if required
  -- must be present if required for verification of message protection
body                 infop (PKIInfoRepContent)
CAProtEncCert        present (object identifier one
                     of PROT_ENC_ALG), with relevant
                     value
  -- to be used if end entity needs to encrypt information for the CA
  -- (e.g., private key for recovery purposes)
SignKeyPairTypes     present, with relevant value
  -- the set of signature algorithm identifiers which this CA will
  -- certify for subject public keys
EncKeypairTypes      present, with relevant value
  -- the set of encryption/key agreement algorithm identifiers which
  -- this CA will certify for subject public keys
PreferredSymmAlg     present (object identifier one
                     of PROT_SYM_ALG) , with relevant
                     value
  -- the symmetric algorithm which this CA expects to be used in later
  -- PKI messages (for encryption)
CAKeyUpdateInfo      optionally present, with
```

```
                     relevant value
  -- the CA may provide information about a relevant root CA key pair
  -- using this field (note that this does not imply that the responding
  -- CA is the root CA in question)
```

```
CurrentCRL           present, with relevant value
  -- the CA may provide a copy of a complete CRL (i.e. fullest possible
  -- one)
protection           present
  -- bits calculated using MSG_MAC_ALG or MSG_SIG_ALG
extraCerts           optionally present
  -- can be used to send some certificates to the end entity. An RA may
  -- add its certificate here.
```

B7. Cross certification (1-way)

Creation of a single cross-certificate (i.e., not two at once). The requesting CA is responsible for publication of the cross-certificate created by the responding CA.

Preconditions:

1. **Responding CA can verify the origin of the request (possibly** requiring out-of-band means) before processing the request.
2. **Requesting CA can authenticate the authenticity of the origin of the** response (possibly requiring out-of-band means) before processing the response

Message Flows:

```
Step#   Requesting CA                                      Responding CA
1       format ccr
2                          ->       ccr       ->
3                                                           handle ccr
4                                                           produce ccp
5                          <-       ccp       <-
6       handle ccp
```

ccr:

```
Field                 Value

sender                Requesting CA name
  -- the name of the CA who produced the message
recipient             Responding CA name
  -- the name of the CA who is being asked to produce a certificate
messageTime           time of production of message
  -- current time at requesting CA
protectionAlg         MSG_SIG_ALG
  -- only signature protection is allowed for this request
senderKID             present if required
  -- must be present if required for verification of message protection
```

```
transactionID         present
  -- implementation-specific value, meaningful to requesting CA.
  -- [If already in use at responding CA then a rejection message
  -- to be produced by responding CA]
```

```
senderNonce            present
  -- 128 (pseudo-)random bits
freeText               any valid value
body                   ccr (CrossCertReqContent)
                       only one FullCertTemplate
                       allowed
  -- if multiple cross certificates are required they must be packaged
  -- in separate PKIMessages
certTemplate           present
  -- details below
version                v1 or v3
  -- <<v3 STRONGLY RECOMMENDED>>
signingAlg             present
  -- the requesting CA must know in advance with which algorithm it
  -- wishes the certificate to be signed


subject                present
  -- may be NULL-DN only if subjectAltNames extension value proposed
validity               present
  -- must be completely specified (i.e., both fields present)
issuer                 present
  -- may be NULL-DN only if issuerAltNames extension value proposed
publicKey              present
  -- the key to be certified which must be for a signing algorithm
extensions             optionally present
  -- a requesting CA must propose values for all extensions which it
  -- requires to be in the cross-certificate
popoSigningKey         present
  -- see "Proof of possession profile" (section B.4)
protection             present
  -- bits calculated using MSG_SIG_ALG
extraCerts             optionally present
  -- can contain certificates usable to verify the protection on
  -- this message


ccp:
Field                  Value

sender                 Responding CA name
  -- the name of the CA who produced the message
recipient              Requesting CA name
  -- the name of the CA who asked for production of a certificate
messageTime            time of production of message
  -- current time at responding CA
protectionAlg          MSG_SIG_ALG
  -- only signature protection is allowed for this message
senderKID              present if required
```

```
  -- must be present if required for verification of message
  -- protection
recipKID              present if required
```

```
transactionID         present
  -- value from corresponding ccr message
senderNonce           present
  -- 128 (pseudo-)random bits
recipNonce            present
  -- senderNonce from corresponding ccr message
freeText              any valid value
body                  ccp (CrossCertRepContent)
                      only one CertResponse allowed
  -- if multiple cross certificates are required they must be packaged
  -- in separate PKIMessages
response              present
status                present
PKIStatusInfo.status  present
  -- if PKIStatusInfo.status is one of:
  --   granted, or
  --   grantedWithMods,
  -- then certifiedKeyPair to be present and failInfo to be absent
failInfo              present depending on
                      PKIStatusInfo.status
  -- if PKIStatusInfo.status is:
  --   rejection
  -- then certifiedKeyPair to be absent and failInfo to be present
  -- and contain appropriate bit settings


certifiedKeyPair      present depending on
                      PKIStatusInfo.status
certificate           present depending on
                      certifiedKeyPair
  -- content of actual certificate must be examined by requesting CA
  -- before publication
protection            present
  -- bits calculated using MSG_SIG_ALG
extraCerts            optionally present
  -- can contain certificates usable to verify the protection on
  -- this message
```

B8. Initial registration / certification

B8.1 Centralised scheme

In this scheme the CA effectively issues a personal security environment (PSE) directly to an end entity using a PKIMessage to transport the resulting certificate, private key, etc.

This profile only allows one certificate and private key to be contained within the PKIMessage.

Preconditions:

1. **The end entity possesses the relevant root CA public key before** processing the PKIMessage.
2. **The end entity is supplied with a symmetric key for decryption of its** private key before processing the PKIMessage.

```
cp:
Field                 Value

sender                CA name
  -- the name of the CA who produced the message
recipient             end entity name
  -- the name of the end entity who is the subject of the certificate
  -- (possibly NULL-DN)
protectionAlg         MSG_SIG_ALG
  -- only signature protection is allowed for this message
senderKID             present if required
  -- must be present if required for verification of message
  -- protection
senderNonce           present
  -- 128 (pseudo-)random bits
freeText              any valid value
body                  cp (CertRepContent)
                      only one CertResponse allowed
  -- if multiple certificates are required they must be packaged in
  -- separate PKIMessages
response              present
status                present
PKIStatusInfo.status  "granted"
  -- no other values allowed (CA must only produce a message if a
  -- certificate has been produced)
certifiedKeyPair      present
certifcate            present
  -- according to pkix-1 profile

privateKey            present
  -- see below
encValue              present
  -- bits of private key encrypted (cleartext bits must be according
  -- to PKCS #1 spec.)
symmAlg               present, SYM_PENC_ALG
  -- algo. to use to decipher encValue using symmetric key distributed
  -- out-of-band
protection            present
  -- bits calculated using MSG_SIG_ALG
```

```
extraCerts             optionally present
  -- can contain certificates usable to verify the protection on
  -- this message
```

B8.2 Basic authenticated scheme

The end entity requests a certificate from a CA. When the CA responds with a message containing a certificate the end entity replies with a confirmation. All messages are authenticated.

This scheme allows the end entity to request certification of a locally-generated public key (typically a signature key). The end entity may also choose to request the centralised generation and certification of another key pair (typically an encryption key pair).

Certification may only be requested for one locally generated public key (for more, use separate PKIMessages).

The end entity must support proof-of-possession of the private key associated with the locally-generated public key.

Preconditions:

1. **The end entity can authenticate the CA s signature based on out-of-**
band means
2. **The end entity and the CA share a symmetric MACing key**

Message flow:

```
Step#    End entity                                      PKI
1        format ir
2                           ->       ir        ->
3                                                        handle ir
4                                                        produce ip
5                           <-       ip        <-
6        handle ip
7        format confirm
8                           ->       conf      ->
9                                                        handle conf
```

For this profile, we mandate that the end entity must include all (i.e. one or two) fullCertTemplates in a single PKIMessage and that the PKI (CA) must produce a single response PKIMessage which contains the complete response (i.e., including the optional second key pair, if it was requested). For simplicity, we also mandate that this message be the final one (i.e. no use of "waiting" status value).

ir:
Field   Value

recipient   CA name
  -- the name of the CA who is being asked to produce a certificate

```
protectionAlg        MSG_MAC_ALG
  -- only MAC protection is allowed for this request, based on
  -- initial authentication key
```

```
senderKID            referenceNum
  -- the reference number which the CA has previously issued to
  -- the end entity (together with the MACing key)
transactionID        present
  -- implementation-specific value, meaningful to end entity.
  -- [If already in use at the CA then a rejection message to be
  -- produced by the CA]
senderNonce          present
  -- 128 (pseudo-)random bits
freeText             any valid value
body                 ir (InitReqContent)
                     only one or two FullCertTemplates
                     are allowed
  -- if more certificates are required requests must be packaged in
  -- separate PKIMessages
protocolEncKey       optionally present.
                     [If present, object identifier
                     must be PROT_ENC_ALG]
  -- if supplied, this short-term asymmetric encryption key (generated
  -- by the end entity) will be used by the CA to encrypt (symmetric
  -- keys used to encrypt) a private key generated by the CA on behalf
  -- of the end entity
fullCertTemplates    one or two present
  -- see below for details, note: fct[0] means the first (which must
  -- be present), fct[1] means the second (which is optional, and used
  -- to ask for a centrally-generated key)
fct[0].              fixed value of zero
   certReqId
  -- this is the index of the template within the message
fct[0].              present
   certTemplate
  -- must include subject public key value, otherwise unconstrained
fct[0].              optionally present if public key
   popoSigningKey    from fct[0].certTemplate is a
                     signing key
  -- proof of possession may be required in this exchange (see section
  -- B.4 for details)
fct[0].              optionally present
   archiveOptions
  -- the end entity may request that the locally-generated private key
  -- be archived
fct[0].              optionally present
   publicationInfo
  -- the end entity may ask for publication of resulting cert.
fct[1].              fixed value of one
   certReqId
  -- the index of the template within the message
```

```
fct[1].               present if protocolEncKey is
   certTemplate       present
  -- must not include actual public key bits, otherwise unconstrained
  -- (e.g., the names need not be the same as in fct[0])
```

```
fct[1].              optionally present
   archiveOptions
fct[1].
   publicationInfo   optionally present
protection           present
  -- bits calculated using MSG_MAC_ALG


ip:
Field                Value

sender               CA name
  -- the name of the CA who produced the message
messageTime          present
  -- time at which CA produced message
protectionAlg        MSG_MAC_ALG
  -- only MAC protection is allowed for this response
recipKID             referenceNum
  -- the reference number which the CA has previously issued to the
  -- end entity (together with the MACing key)
transactionID        present
  -- value from corresponding ir message
senderNonce          present
  -- 128 (pseudo-)random bits
recipNonce           present
  -- value from senderNonce in corresponding ir message
freeText             any valid value
body                 ir (CertRepContent)
                     contains exactly one response
                     for each request
  -- The PKI (CA) responds to either one or two requests as appropriate.
  -- crc[0] denotes the first (always present); crc[1] denotes the
  -- second (only present if the ir message contained two requests).
crc[0].              fixed value of zero
   certReqId
  -- must contain the response to the first request in the corresponding
  -- ir message
crc[0].status.       present, positive values allowed:
   status               "granted", "grantedWithMods"
                     negative values allowed:
                        "rejection"
crc[0].status.       present if and only if
   failInfo          crc[0].status.status is "rejection"
crc[0].              present if and only if
   certifiedKeyPair  crc[0].status.status is
                        "granted" or "grantedWithMods"
certificate          present unless end entity s public
                     key is an encryption key and POP
```

is required by CA/RA

encryptedCert   present if and only if end entity s
                     public key is an encryption key
                     and POP is required by CA/RA

```
publicationInfo      optionally present
  -- indicates where certificate has been published (present at
  -- discretion of CA)
crc[1].              fixed value of one
   certReqId
  -- must contain the response to the second request in the
  -- corresponding ir message


crc[1].status.       present, positive values allowed:
   status               "granted", "grantedWithMods"
                     negative values allowed:
                        "rejection"
crc[1].status.       present if and only if
   failInfo          crc[0].status.status is "rejection"
crc[1].              present if and only if
   certifiedKeyPair  crc[0].status.status is "granted"
                     or "grantedWithMods"
certificate          present
privateKey           present
publicationInfo      optionally present
  -- indicates where certificate has been published (present at
  -- discretion of CA)
protection           present
  -- bits calculated using MSG_MAC_ALG
extraCerts           optionally present
  -- the CA may provide additional certificates to the end entity



conf:
Field                Value

recipient            CA name
  -- the name of the CA who was asked to produce a certificate
transactionID        present
  -- value from corresponding ir and ip messages
senderNonce          present
  -- value from recipNonce in corresponding ir message
recipNonce           present
  -- value from senderNonce in corresponding ip message
protectionAlg        MSG_MAC_ALG
  -- only MAC protection is allowed for this request.  The MAC is
  -- based on the initial authentication key if only a signing key
  -- pair has been sent in ir for certification or if POP is not
  -- required by CA/RA.  Otherwise, the MAC is based on a key derived
  -- from the symmetric key used to decrypt the returned encryptedCert.
senderKID            referenceNum
  -- the reference number which the CA has previously issued to the
```

```
  -- end entity (together with the MACing key)
body                 conf (PKIConfirmContent)
  -- this is an ASN.1 NULL
protection           present
  -- bits calculated using MSG_MAC_ALG
```

Appendix C: "Compilable" ASN.1 Module

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

  PKIHeader ::= SEQUENCE {
      pvno                INTEGER     { ietf-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)
  }

  PKIFreeText ::= CHOICE {
      iA5String  [0] IA5String,
      bMPString  [1] BMPString
  } -- note that the text included here would ideally be in the
    -- preferred language of the recipient
```

```
PKIBody ::= CHOICE {       -- message-specific body elements
    ir      [0]  InitReqContent,
    ip      [1]  InitRepContent,
    cr      [2]  CertReqContent,
    cp      [3]  CertRepContent,
    p10cr   [4]  PKCS10CertReqContent, -- imported from [PKCS10]
    popdecc [5]  POPODecKeyChallContent,
    popdecr [6]  POPODecKeyRespContent,
    kur     [7]  KeyUpdReqContent,
    kup     [8]  KeyUpdRepContent,
    krr     [9]  KeyRecReqContent,
    krp     [10] KeyRecRepContent,
    rr      [11] RevReqContent,
    rp      [12] RevRepContent,
    ccr     [13] CrossCertReqContent,
    ccp     [14] CrossCertRepContent,
    ckuann  [15] CAKeyUpdAnnContent,
    cann    [16] CertAnnContent,
    rann    [17] RevAnnContent,
    crlann  [18] CRLAnnContent,
    conf    [19] PKIConfirmContent,
    nested  [20] NestedMessageContent,
    infor   [21] PKIInfoReqContent,
    infop   [22] PKIInfoRepContent,
    error   [23] ErrorMsgContent
}

PKIProtection ::= BIT STRING

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

PasswordBasedMac ::= OBJECT IDENTIFIER

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

DHBasedMac ::= OBJECT IDENTIFIER

DHBMParameter ::= SEQUENCE {
```

```
      owf                 AlgorithmIdentifier,
      -- AlgId for a One-Way Function (SHA-1 recommended)
      mac                 AlgorithmIdentifier
      -- the MAC AlgId (e.g., DES-MAC or Triple-DES-MAC [PKCS #11])
  }
```

```
NestedMessageContent ::= ANY
-- This will be a PKIMessage


CertTemplate ::= SEQUENCE {
    version    [0] Version               OPTIONAL,
    -- used to ask for a particular syntax version
    serial     [1] INTEGER               OPTIONAL,
    -- used to ask for a particular serial number
    signingAlg [2] AlgorithmIdentifier   OPTIONAL,
    -- used to ask the CA to use this alg. for signing the cert
    subject    [3] Name                  OPTIONAL,
    validity   [4] OptionalValidity      OPTIONAL,
    issuer     [5] Name                  OPTIONAL,
    publicKey  [6] SubjectPublicKeyInfo  OPTIONAL,
    issuerUID  [7] UniqueIdentifier      OPTIONAL,
    subjectUID [8] UniqueIdentifier      OPTIONAL,
    extensions [9] Extensions            OPTIONAL
    -- the extensions which the requester would like in the cert.
}

OptionalValidity ::= SEQUENCE {
    notBefore  [0] UTCTime OPTIONAL,
    notAfter   [1] UTCTime OPTIONAL
}

EncryptedValue ::= SEQUENCE {
    encValue          BIT STRING,
    -- the encrypted value itself
    intendedAlg   [0] AlgorithmIdentifier  OPTIONAL,
    -- the intended algorithm for which the value will be used
    symmAlg       [1] AlgorithmIdentifier  OPTIONAL,
    -- the symmetric algorithm used to encrypt the value
    encSymmKey    [2] BIT STRING           OPTIONAL,
    -- the (encrypted) symmetric key used to encrypt the value
    keyAlg        [3] AlgorithmIdentifier  OPTIONAL
    -- algorithm used to encrypt the symmetric key
}
```

```
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
    -- FullCertTemplate
}

PKIFailureInfo ::= BIT STRING {
-- since we 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
 }
```

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

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.
}

PKIArchiveOptions ::= CHOICE {
    encryptedPrivKey     [0] EncryptedValue,
    -- the actual value of the private key
    keyGenParameters     [1] KeyGenParameters,
    -- parameters which allow the private key to be re-generated
    archiveRemGenPrivKey [2] BOOLEAN
    -- set to TRUE if sender wishes receiver to archive the private
    -- key of a key pair which the receiver generates in response to
    -- this request; set to FALSE if no archival is desired.
}

KeyGenParameters ::= OCTET STRING
    -- an alternative to sending the key is to send the information
    -- about how to re-generate the key (e.g. for many RSA
    -- implementations one could send the first random number tested
    -- for primality).
    -- The actual syntax for this parameter may be defined in a
    -- subsequent version of this document or in another standard.

PKIPublicationInfo ::= SEQUENCE {
   action     INTEGER {
                dontPublish (0),
                pleasePublish (1)
              },
   pubInfos  SEQUENCE OF SinglePubInfo OPTIONAL
     -- pubInfos must 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)
      },
      pubLocation   GeneralName OPTIONAL
  }
```

```
  FullCertTemplates ::= SEQUENCE OF FullCertTemplate

  FullCertTemplate ::= SEQUENCE {
      certReqId              INTEGER,
      -- to match this request with corresponding response
      -- (note:  must be unique over all FullCertReqs in this message)
      certTemplate           CertTemplate,
      popoSigningKey     [0] POPOSigningKey      OPTIONAL,
      archiveOptions     [1] PKIArchiveOptions   OPTIONAL,
      publicationInfo    [2] PKIPublicationInfo  OPTIONAL,
      oldCertId          [3] CertId              OPTIONAL
      -- id. of cert. which is being updated by this one
  }

  POPOSigningKey ::= SEQUENCE {
      poposkInput          POPOSKInput,
      alg                  AlgorithmIdentifier,
      signature            BIT STRING
      -- the signature (using "alg") on the DER-encoded
      -- value of poposkInput
  }

  POPOSKInput ::= CHOICE {
      popoSigningKeyInput      [0] POPOSigningKeyInput,
      certificationRequestInfo     CertificationRequestInfo
      -- imported from [PKCS10] (note that if this choice is used,
      -- POPOSigningKey is simply a standard PKCS #10 request; this
      -- allows a bare PKCS #10 request to be augmented with other
      -- desired information in the FullCertTemplate before being
      -- sent to the CA/RA)
  }

  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] BIT STRING
          -- used if no authenticated GeneralName currently exists for
          -- the sender; publicKeyMAC contains a password-based MAC
          -- (using the protectionAlg AlgId from PKIHeader) on the
          -- DER-encoded value of publicKey
      },
      publicKey            SubjectPublicKeyInfo    -- from CertTemplate
  }

  InitReqContent ::= SEQUENCE {
```

```
     protocolEncKey      [0] SubjectPublicKeyInfo  OPTIONAL,
     fullCertTemplates       FullCertTemplates
 }

 InitRepContent ::= CertRepContent
```

```
  CertReqContent ::= CHOICE {
      fullCertTemplates    [0] FullCertTemplates,
      pkcs10CertReqContent [1] PKCS10CertReqContent
  }

  POPODecKeyChallContent ::= SEQUENCE OF Challenge
  -- One Challenge per encryption key certification request (in the
  -- same order as these requests appear in FullCertTemplates).

  Challenge ::= SEQUENCE {
      owf                 AlgorithmIdentifier  OPTIONAL,
      -- must be present in the first Challenge; may be omitted in any
      -- subsequent Challenge in POPODecKeyChallContent (if omitted,
      -- then the owf used in the immediately preceding Challenge is
      -- to be used).
      witness             OCTET STRING,
      -- the result of applying the one-way function (owf) to a
      -- randomly-generated INTEGER, A.  [Note that a different
      -- INTEGER must be used for each Challenge.]
      challenge           OCTET STRING
      -- the encryption (under the public key for which the cert.
      -- request is being made) of Rand, where Rand is specified as
      --   Rand ::= SEQUENCE {
      --      int      INTEGER,
      --       - the randomly-generated INTEGER A (above)
      --      sender   GeneralName
      --       - the sender's name (as included in PKIHeader)
      --   }
  }

  POPODecKeyRespContent ::= SEQUENCE OF INTEGER
  -- One INTEGER per encryption key certification request (in the
  -- same order as these requests appear in FullCertTemplates).  The
  -- retrieved INTEGER A (above) is returned to the sender of the
  -- corresponding Challenge.

  CertRepContent ::= SEQUENCE {
      caPub           [1] Certificate             OPTIONAL,
      response            SEQUENCE OF CertResponse
  }

  CertResponse ::= SEQUENCE {
      certReqId           INTEGER,
      -- to match this response with corresponding request
      status              PKIStatusInfo,
      certifiedKeyPair    CertifiedKeyPair    OPTIONAL
  }

  CertifiedKeyPair ::= SEQUENCE {
```

```
      certificate     [0] Certificate         OPTIONAL,
      encryptedCert   [1] EncryptedValue      OPTIONAL,
      privateKey      [2] EncryptedValue      OPTIONAL,
      publicationInfo [3] PKIPublicationInfo  OPTIONAL
  }
```

```
KeyUpdReqContent ::= SEQUENCE {
    protocolEncKey      [0] SubjectPublicKeyInfo  OPTIONAL,
    fullCertTemplates   [1] FullCertTemplates     OPTIONAL
}

KeyUpdRepContent ::= InitRepContent

KeyRecReqContent ::= InitReqContent

KeyRecRepContent ::= SEQUENCE {
    status                  PKIStatusInfo,
    newSigCert          [0] Certificate                   OPTIONAL,
    caCerts             [1] SEQUENCE OF Certificate       OPTIONAL,
    keyPairHist         [2] SEQUENCE 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,
    -- from the DAM, so that CA knows which Dist. point to use
    badSinceDate        GeneralizedTime  OPTIONAL,
    -- indicates best knowledge of sender
    crlEntryDetails     Extensions
    -- requested crlEntryExtensions
}

RevRepContent ::= SEQUENCE {
    status              PKIStatusInfo,
    revCerts        [0] SEQUENCE OF CertId OPTIONAL,
    -- identifies the certs for which revocation was requested
    crls            [1] SEQUENCE OF CertificateList  OPTIONAL
    -- the resulting CRLs (there may be more than one)
}

CrossCertReqContent ::= CertReqContent

CrossCertRepContent ::= CertRepContent

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               OBJECT IDENTIFIER,
      infoValue              ANY DEFINED BY infoType  OPTIONAL
  }
  -- Example InfoTypeAndValue contents include, but are not limited to:
  --   { CAProtEncCert    = { xx }, Certificate                      }
  --   { SignKeyPairTypes = { xx }, SEQUENCE OF AlgorithmIdentifier }
  --   { EncKeyPairTypes  = { xx }, SEQUENCE OF AlgorithmIdentifier }
  --   { PreferredSymmAlg = { xx }, AlgorithmIdentifier              }
  --   { CAKeyUpdateInfo  = { xx }, CAKeyUpdAnnContent               }
  --   { CurrentCRL       = { xx }, CertificateList                  }


  PKIInfoReqContent ::= SET OF InfoTypeAndValue
  -- The OPTIONAL infoValue parameter of InfoTypeAndValue is unused.
  -- The CA is free to ignore any contained OBJ. IDs that it does not
  -- recognize.
  -- The empty set indicates that the CA may send any/all information
  -- that it wishes.

  PKIInfoRepContent ::= SET OF InfoTypeAndValue
  -- The end entity is free to ignore any contained OBJ. IDs that it
  -- does not recognize.

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