Government of Canada
Public Key Infrastructure
(GoC PKI)

GOC PKI Certificate and Key Management Interface Specification

Version 1.0
March 2000

Table of Contents

List of figures

1 Overview of PKI management

1.1 Introduction

This section describes a model for a Public Key Infrastructure (PKI) to which the Government of Canada Public Key Infrastructure (GOC PKI) conforms, including the following PKI components:

- End-entities (EEs);
- Certification Authorities (CAs);
- Registration Authorities (RAs);
- Time Stamp server; and
- PKI Repository (Directory).

Management protocols are required to support on-line interactions between these 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 that issue cross-certificates for each other.

1.2 Definitions of PKI entities

The entities involved in PKI management include the EE (i.e., the entity to be named in the subject field of a certificate) and the CA (i.e., the entity named in the issuer field of a certificate). An RA may also be involved in PKI management.

1.2.1 Subjects and End Entities

The term "subject" is used to refer to the entity named in the subject field of a certificate. When distinguishing the tools and/or software used by the subject (e.g., a local certificate management module), the term "subject equipment" is used.

It is important to note that the end entities here will include not only human users of applications, but also applications themselves. This factor influences the protocols that the PKI management operations use (e.g., application 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 entity; 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).

1.2.2 Certification Authority

The CA may or may not actually be a real "third party" from the end-entity's point of view. Often, the CA will actually belong to the same organization as the end entities it supports.

The term CA is used 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, the term "CA equipment" is used.

The term "root CA" is used to indicate a CA that is directly trusted by an end-entity. This term is not meant to imply that a root CA is necessarily at the top of any hierarchy, simply that the CA in question is trusted directly. A "subordinate CA" is one that is not a root CA for the end-entity in question.

1.2.3 Registration Authority

In addition to end-entities and CAs, many environments call for the existence of an RA separate from the CA. The functions which the RA 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, etc.

This document views the RA as an optional component. When the RA 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. An RA is itself an end-entity.

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

1.2.4 Time Stamp Authority

Time Stamp Protocols, Section 7, Reference 1, describes the format of the data returned by a Time Stamp Authority (TSA) and the protocols to be used when communicating with it. The time stamping service can be used as a Trusted Third Party (TTP) as one component in building reliable non-repudiation services. The TSA provides a "proof-of-existence" for a particular datum at an instant in time. That is, the TSA is a TTP that creates time stamp tokens in order to indicate that a datum existed at a particular point in time. A TSA may also be used when a trusted time reference is required and when the local clock available cannot be trusted by all parties.

The TSA's role is to time stamp a datum to establish evidence indicating the time at which the datum existed. This can then be used, for example, to verify that a digital signature was applied to a message before the corresponding certificate was revoked thus allowing a revoked public key certificate to be used for verifying signatures created prior to the time of revocation. This is an important public key infrastructure operation. The TSA can also be used to indicate the time of submission when a deadline is critical, or to indicate the time of transaction for entries in a log.

1.2.5 PKI Repository

The PKI Repository, or Directory, can be any directory service that communicates using the Lightweight Directory Access Protocol (LDAP). The Directory contains an entry for each person in the organization. The Directory also serves as a repository for user’s encryption public key certificates. When end-entity certificates are generated, they are written into the Directory.

Storing encryption certificates in the Directory allows end entities to access the trustworthy encryption public keys of other end entities and encrypt information for them. The Directory keeps lists of revoked certificates in addition to a list of valid encryption public key certificates that are owned by legitimate end entities. These lists are known as Certificate Revocation Lists (CRLs). In cross-certified systems, the Directory also stores cross-certificates and Authority Revocation Lists (ARLs). When a cross-certificate is revoked, it is noted on an ARL.

1.3 PKI Management Requirements

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

1) PKI management must conform to the ISO 9594-8 standard, Section 7, Reference 2, and the associated amendments (certificate extensions).

2) PKI management must conform to the other parts of the PKIX series of documents.

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, (e.g., RSA, DSA, MD5, SHA-1). This means that any given CA, RA, or end-entity may, in principle, 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.

7) PKI management protocols must support the publication of certificates by the end-entity concerned, by an RA, or by a CA.

8) PKI management protocols must support the production of Certificate Revocation Lists (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, (e.g., TCP/IP).

10) Final authority for certification 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. 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 (e.g., a CA may shorten the validity period requested).

11) A graceful, scheduled changeover from one non-compromised CA key pair to the next (CA key update) must be supported. If the CA key is compromised, re-initialization must be performed for all entities in the domain of that CA. An end-entity who has a copy of 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. This is 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 regardless of whether the communication is with an RA or CA.

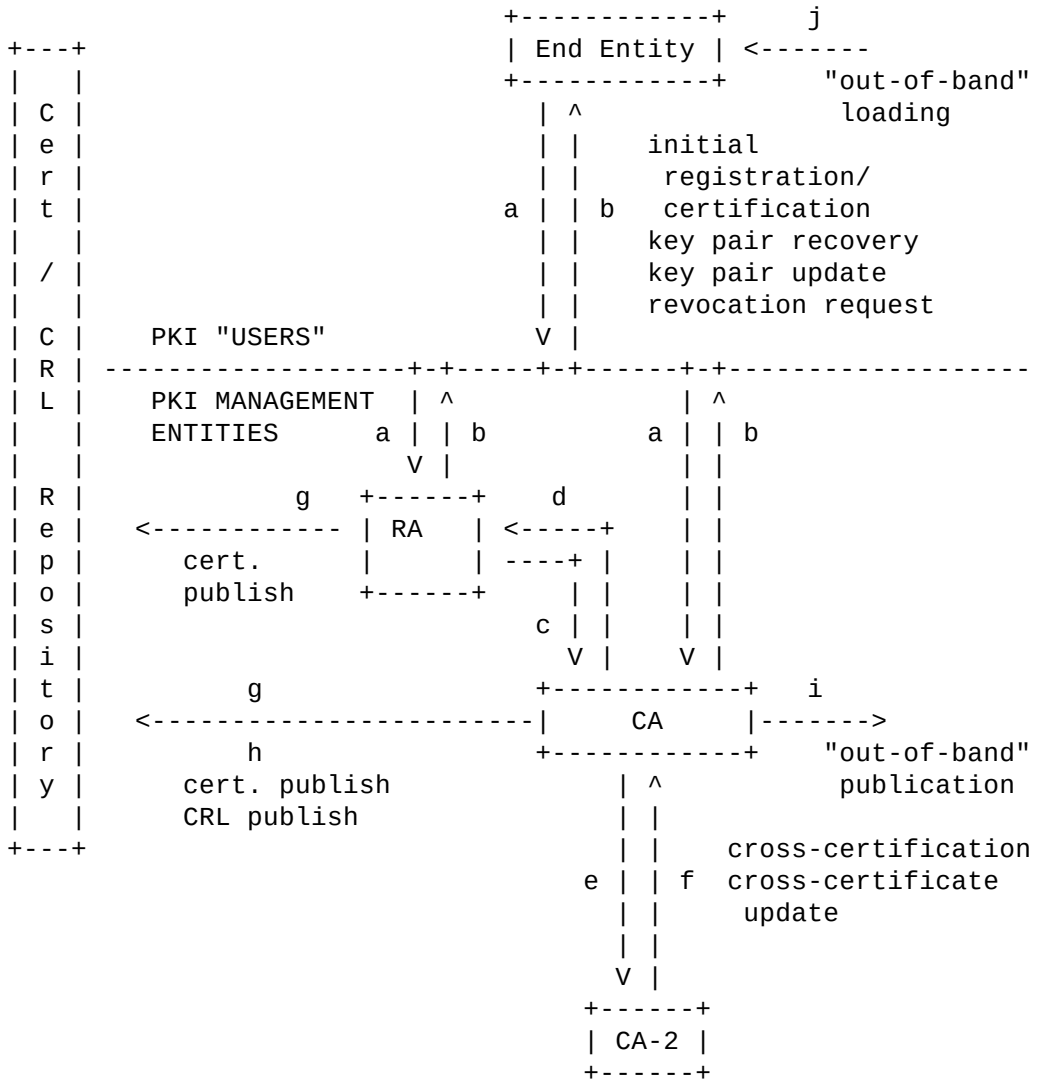
13) Where an end-entity requests a certificate containing a given public key value, the end-entity must be ready to demonstrate possession of the corresponding private

key value. This may be accomplished in various ways, depending on the type of certification request.

1.4 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.

Figure 1. PKI Entities and Management Operations



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) **Certification**: Various operations result in the creation of new certificates:

a) **Initialization (End-entity initialization and initial registration/certification)**: This includes importing a root CA public key and requesting information about the options supported by a PKI management entity. This is the process whereby an end-entity first makes itself known to a CA or RA, prior to the CA issuing a certificate or certificates for that end-entity. The end result of this process (when it is successful) is that a CA issues a certificate for an end-entity's public key, and returns that certificate to the end-entity and/or posts that certificate in a public repository.

b) **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.

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

d) **Cross-certification**: One CA requests issuance of a cross-certificate from another CA. A "cross-certificate" is a certificate in which the subject CA and the issuer CA are distinct and **SubjectPublicKeyInfo** contains a verification key (i.e., the certificate has been issued for the subject CA's signing key pair). When it is necessary to distinguish more finely, the following terms may be used: a cross-certificate is called an "inter-domain cross-certificate" if the subject and issuer CAs belong to different administrative domains; it is called an "intra-domain cross-certificate" otherwise.

Note: The above definition of "cross-certificate" aligns with the defined term "CA-certificate" in X.509.

Note: In many environments the term "cross-certificate", unless further qualified, will be understood to be synonymous with "inter-domain cross-certificate" as defined above.

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

a) **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 methods (LDAP, for example) as described in the "Operational Protocols" documents of the PKIX series of specifications.

b) **CRL publication**: As for certificate publication.

4) Recovery operations: some PKI management operations are used when an end-entity has "lost" its credentials (e.g., password, private key).

a) **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 an entity 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.

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

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

2 Subscriber and CA Registration Models

This section describes the accepted models for subscriber and CA registration in a PKI and indicates how the qualifications and identity of subscribers and subject CAs must be confirmed and how the certificate requester must be correctly authenticated in the certificate request message.

2.1 Assumptions and Restrictions

2.1.1 Initialization (initialization/registration/certification)

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

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

We can however, classify the initial registration/certification schemes that 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 that 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 that will arise in real use, while the optional schemes are available for special cases that arise less frequently. In this way we achieve a balance between flexibility and ease of implementation.

2.1.2 Criteria used

2.1.2.1 Initialization (initialization/initial registration/certification)

In terms of the PKI messages that 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, a RA, or a CA.

2.1.2.2 End-entity message origin authentication

The on-line messages produced by the end-entity who 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's PKI messages are authenticated or not.

Note: The authentication of the PKI is an important issue and can be achieved in various models. The most secure method is via an on-line delivery of the CA public key using a secure protocol with a shared secret. A less secure method is simply once the root-CA public key has been installed at the end-entity's equipment via the equipment vendor.

Note: An initialization procedure can be secure where the messages from the end-entity and the PKI are authenticated via some out-of-band means.

2.1.2.3 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 **PKIMessage** (refer to Section 3.1). Note that this does not preclude a centralized key generation service - the actual key pair may have been generated elsewhere and transported to the end-entity, RA, or CA using a (proprietary or standardized) key generation request/response protocol (outside the scope of this specification).

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

2.1.2.4 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.

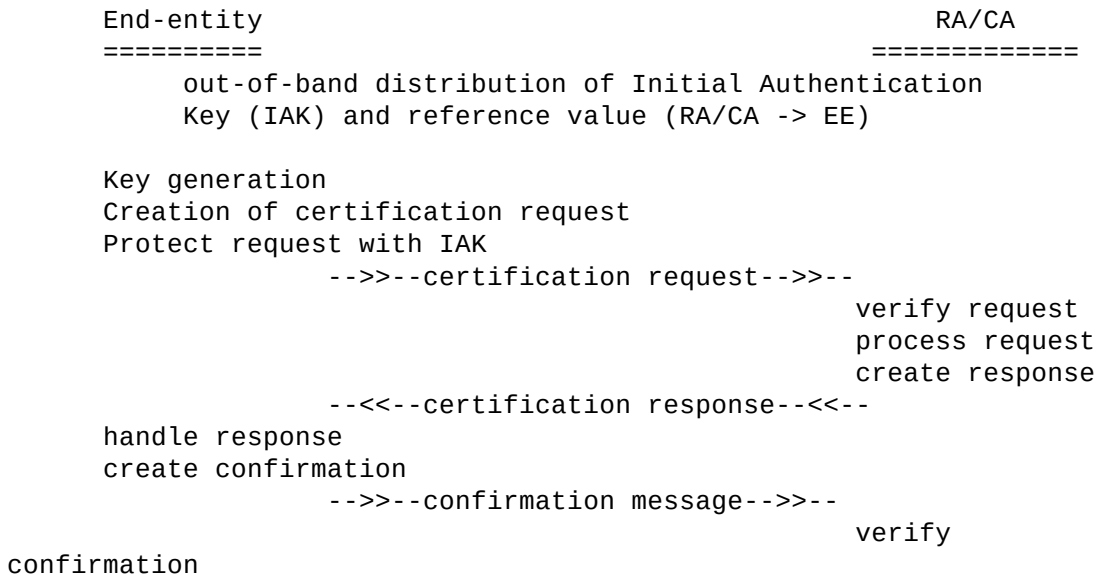
2.1.2.5 Basic authenticated scheme

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 basic authenticated scheme is as depicted in Figure 2:

Figure 2. Basic Authenticated Scheme



Where verification of the confirmation message fails, the RA/CA must revoke the newly issued certificate if it has been published or otherwise made available.

2.2 Proof of Possession (POP) of Private Key

In order to prevent certain attacks and to allow a CA/RA to properly check the validity of the binding between an end-entity and a 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 how to enforce POP (e.g., out-of-band procedural means versus PKIX-CMP in-band messages) in its certification exchanges (i.e., this may be a policy issue).

However, it is required that CAs/RAs must enforce POP by some means because there are currently many non-PKIX operational protocols in use that do not explicitly check the binding between the end-entity and the private key.

2.2.1 Signature Keys

The signature keys, the client will sign a value to prove possession of the private key. This value is contained in the certificate request.

2.2.2 Encryption Keys

For encryption keys, the client can provide the private key to the CA or, be required to decrypt a value in order to prove possession of the private key. The CA will support the indirect method which is for the CA to issue a certificate encrypted for the client and have the client demonstrate it's ability to decrypt this certificate in a confirmation message.

This allows a CA to issue a certificate in a form that can only be used by the intended end-entity. This specification encourages use of 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).

3 PKIX-CMP management protocol

The IETF PKIX RFC 2510 (Certificate Management Protocols), Section 7, Reference 3, defines the protocol between a CA and a client. The GOC PKI supports a subset of this protocol and uses it to provide certificate services to clients. As of the June 2000, all new clients will be managed using PKIX-CMP.

3.1 Overview

This section provides an overview of the PKI management process using the PKIX-CMP standard.

As described in Section 1.4, the PKIX-CMP standard defines message and message syntax to perform the various PKI management operations, including the following:

1) Initialization (**initReq**, **initRep**)
2) Certificate Request (**certReq**, **certRep**)
3) Cross-Certification (**crossCertReq**, **crossCertRep**)
4) Key Update (**keyUpdReq**, **keyUpdRep**)
5) Revocation (**revReq**, **revRep**)
6) Key Recovery (**keyRecReq** , **keyRecRep**)
7) PKIX confirmation message (**pKIConfirm**)
8) PKIX general message (**genM**, **genP**)

3.1.1 Messaging format and protocol

All of the messages used in this specification for the purposes of PKI management use the following structure:

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

The **PKIHeader** contains information which is common to many PKI messages. It contains some header information for addressing and transaction identification.

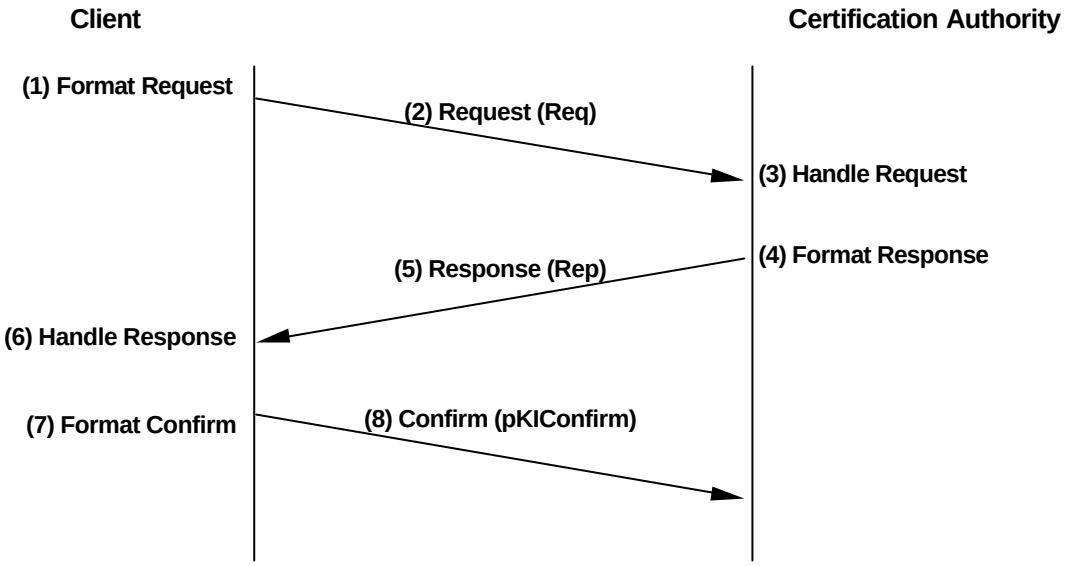
The **PKIBody** contains message-specific information (e.g., **CertReqMessages**, **CertRepMessage**).

The **PKIProtection**, when used, contains bits that protect the PKI message. Depending on the circumstances, the **PKIProtection** bits may contain a MAC or a digital signature. The protection applied to particular PKIX messages is described in Section 3.1.5.

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

The general PKIX protocol behavior is as illustrated in Figure 3.

Figure 3: PKIX protocol



All of the supported PKI management functions (transactions) are initiated by the client, and all consist of a request (client to CA) followed by a response (CA to client) followed by a confirmation (client to CA). The last request is used as a final acknowledgment to CA (**pKIConfirm**).

All PKI Management functions that result in certificate generation by CA require that the client send a confirmation message upon acceptance of the newly issued certificate. If a client chooses to reject a certificate response from CA, it may either:

- send no confirmation which will result in an immediate revocation of those certificates issued by CA; or
- return an Error Message containing failure information useful for human consumption (i.e., for the log files at CA).

All requests for certificates, whether the first time initialization or not, will require that proof-of-possession be established by CA for both signing keys and encryption keys.

For signature keys, the client will sign a value to prove possession of the private key. This value is contained in the certificate request.

For encryption keys, the client can provide the private key to CA or, be required to decrypt a value in order to prove possession of the private key. CA will support the indirect method which is for the CA to issue a certificate encrypted for the client and have the client demonstrate its ability to decrypt this certificate in a confirmation message.

3.1.2 PKI 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.

The following data structure is used to contain this information:

```
PKIHeader ::= SEQUENCE {
        pvno            INTEGER                 { ietf-version2 (1) },
        -- currently 1
        sender                  GeneralName,
        -- identifies the sender
        recipient               GeneralName,
        -- identifies the intended recipient
        messageTime  [0] GeneralizedTime    OPTIONAL,
        -- time of production of this message
        protectionAlg           [1] AlgorithmIdentifier OPTIONAL,
        -- algorithm used for calculation of protection bits
        senderKID       [2] KeyIdentifier           OPTIONAL,
        -- to identify specific keys used for protection (reference number)
        recipKID        [3] KeyIdentifier           OPTIONAL,
        -- to identify specific keys used for protection (reference number)
        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,
        -- inserted by the creator of this message to provide replay protection
        recipNonce      [6] OCTET STRING            OPTIONAL,
        -- a nonce previously inserted in a related message by the intended recipient of
this    -- message to provide replay protection
        freeText        [7] PKIFreeText             OPTIONAL,
        -- this field is intended for human consumption
        generalInfo     [8] SEQUENCE SIZE (1..MAX) OF InfoTypeAndValue
                                        OPTIONAL
        -- this may be used to convey context-specific information }

PKIFreeText ::= SEQUENCE SIZE (1..MAX) OF UTF8String
        -- text encoded as UTF-8 String

AlgorithmIdentifier  ::=  SEQUENCE  {
        Algorithm               OBJECT IDENTIFIER,
        Parameters              ANY DEFINED BY algorithm OPTIONAL  }
```

The fields of the **PKIHeader** data structure are described below:

- The **pvno** field is fixed (at one) for the current version of the PKIX-CMP specification.
- The sender field contains the name of the sender of the **PKIMessage**. This name (in conjunction with **senderKID**, if supplied) should be able to verify the protection on the message. If nothing about the sender is known to the sending entity (e.g., in the **initReq** message, where the end-entity may not know its own Distinguished Name (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 (note that **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.
- The **transactionID** field within the message header may be used to allow the recipient of a response message to 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 (in any number of languages). The first language used in this sequence indicates the desired language for replies.
- The **generalInfo** field may be used to send machine-processable additional data to the recipient.

3.1.3 PKI Body

The **PKIBody** contains message-specific information, and varies depending on the PKI management operation in question.

```
PKIBody ::= CHOICE {
      ir       [0]  CertReqMessages,        --Initialization Request
      ip       [1]  CertRepMessage,         --Initialization Response
      cr       [2]  CertReqMessages,        --Certification Request
      cp       [3]  CertRepMessage,         --Certification Response
      p10cr    [4]  CertificationRequest,   --PKCS #10 Cert. Req.
      kur      [7]  CertReqMessages,        --Key Update Request
      kup      [8]  CertRepMessage,         --Key Update Response
      krr      [9]  CertReqMessages,        --Key Recovery Request
      krp      [10] KeyRecRepContent,       --Key Recovery Response
      rr       [11] RevReqContent,          --Revocation Request
      rp       [12] RevRepContent,          --Revocation Response
      ccr      [13] CertReqMessages,        --Cross-Cert. Request
      ccp      [14] CertRepMessage,         --Cross-Cert. Response
      conf     [19] PKIConfirmContent,      --Confirmation
      genm          [21] GenMsgContent,     --General Message
      genp          [22] GenRepContent,     --General Response
      error    [23] ErrorMsgContent         --Error Message }
```

3.1.4 PKI Message Protection

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

When protection is applied the following structure is used:

PKIProtection ::= BIT STRING

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

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

Depending on the circumstances the **PKIProtection** bits may contain a Message Authentication Code (MAC) or signature.

3.1.4.1 MAC-based protection

In the case where the sender and recipient share secret information (established via out-of-band means or from a previous PKI management operation), **PKIProtection** will contain a MAC value and the **protectionAlg** (**MSG_MAC_ALG**) will be the following:

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

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., CAST5-MAC) }

AlgorithmIdentifier  ::=  SEQUENCE  {

    Algorithm           OBJECT IDENTIFIER,
    Parameters          ANY DEFINED BY algorithm OPTIONAL  }
```

3.1.4.2 Signature-based protection

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

3.1.5 Message Origin Authentication

All messages from clients must contain origin authentication.

The methods of authentication depend on the operation in question. There are two supported possibilities:

- MACs
- digital signatures

3.1.5.1 Password-based MAC

In this case, all messages involved in PKI Management will be authenticated using the Basic Authenticated Scheme. Message origin authentication is achieved by the CA issuing the client a reference number (used for identification) and a secret value or authentication code (used to generate a shared secret key) which will provide the message origin authentication when used to protect PKI messages involved in the operation. This information is obtained out-of-band.

The CA protects the messages in the same way as the client, thus authenticating the PKI to the client. The following operations will use this scheme for message authentication:

- General Message
- Initialization
- Key Recovery
- Cross Certification

3.1.5.2 Digital Signature

In cases where an existing client already has a valid signing key pair (and thus message origin authentication is implicitly available), messages may be authenticated via digital signature. This would typically occur with the following messages:

- General Message
- Certification
- Key Update
- Revocation

3.1.6 Proof of Possession

All requests for certificates (whether it be first time initialization, update etc.) will require that proof-of-possession be established by the CA for both signing keys and encryption keys.

For signature keys, the client will sign a value to prove possession of the private key. This value is contained in the certificate request. For encryption keys, the client can provide the private key to the CA or be required to decrypt a value in order to prove possession of the private key. The CA will support the indirect method which is for the CA to issue a certificate encrypted for the client and have the client demonstrate it's ability to decrypt this certificate in a confirmation message.

3.1.7 Protection of Private Keys

In certain circumstances, private key information is passed between the client and the CA. An example of this is when the CA creates the encryption key pair and sends the private key back to the client.

Generally speaking, the end point (client or CA) which is to send the private key information must obtain a protocol encryption key from the other end point. This key will be used to protect a symmetric key that in turn is used to protect the private key component in question.

3.1.8 Algorithms

3.1.8.1 Digital signature algorithms

The PKIX-CMP signature algorithm choices for the client are: RSA-512, RSA-1024 (default), RSA-2048, DSA-1024, and ECDSA-192.

The PKIX-CMP signature algorithm choices for the CA are: RSA-1024 (default), RSA-2048, and DSA-1024.

For PKIX-CMP interoperability, support for DSA/SHA-1 is mandatory. This applies to the **MSG_SIG_ALG** field. RSA/MD5 is an alternative to the mandatory **AlgorithmIdentifier**.

3.1.8.2 Message Authentication Code algorithms

The PKIX-CMP MAC algorithm choices for the client are: triple-DES, CAST5-128 (default), and HMAC-SHA-1.

The PKIX-CMP MAC algorithm choice for the CA can only be HMAC-SHA-1.

For PKIX-CMP interoperability, support for **PasswordBasedMAC** is mandatory. This applies to the **MSG_MAC_ALG** field. HMAC is an alternative to the mandatory **AlgorithmIdentifier**.

3.1.8.3 Encryption algorithms

The ephemeral encryption key choices for the client are: RSA 1024 (default) and RSA 2048. This key is encrypted with a symmetric key. The symmetric encryption algorithm choices for the client are: CAST5-128 (default) and triple-DES.

The ephemeral encryption algorithm choices for the CA are: RSA 1024 (default) and RSA 2048. This key is encrypted with a symmetric key. The symmetric encryption algorithm choices for the CA are: CAST5-128 (default) and triple-DES.

For PKIX-CMP interoperability, support for triple-DES by the client and the CA is mandatory. This applies to the **PROT_SYM_ALG** field. RC5, CAST5-128, and others are alternatives to the mandatory **AlgorithmIdentifier**.

3.1.8.4 Mandatory algorithm identifiers and specifications

3.1.8.4.1 DSA/SHA-1

- AlgId: {1 2 840 10040 4 3};
- NIST, FIPS PUB 186: Digital Signature Standard, 1994; and
- Public Modulus size: 1024 bits.

3.1.8.4.2 PasswordBasedMAC

- {1 2 840 113533 7 66 13}, with SHA-1 {1 3 14 3 2 26} as the **owf** parameter and HMAC-SHA1 {1 3 6 1 5 5 8 1 2} as the **mac** parameter;
- NIST, FIPS PUB 180-1: Secure Hash Standard, April 1995; and
- H. Krawczyk, M. Bellare, R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", Internet Request for Comments 2104, February 1997.

3.1.8.4.3 Triple-DES

- {1 2 840 113549 3 7} (used in RSA's BSAFE and in S/MIME).

3.2 General Message

A general message exchange may first need to take place prior to any other PKI management function. In certain cases it may be necessary for the client to obtain information from the CA before performing PKI management functions. For example, if a client is requesting an encryption certificate for a client-generated key pair with backup, it must obtain a protocol encryption certificate from the CA in order to protect the private portion of that key pair. This is accomplished through a General Message exchange.

Currently, a client may request a policy certificate (Section 3.2.1.1 and Section 3.2.2.1) or a CA protocol encryption certificate (Section 3.2..1.2 and Section 3.2.2.2) via a PKIX general message (**genM**). The client will send a general message to the CA requesting details that will be required for later PKI management operations (i.e., protocol encryption certificate or policy certificate). The response from the CA must be the **genP** message. A **pKIConfirm** message is not required from the client.

The General Message exchange consists of two messages:

- General Message Request (**genM**), and
- General Message Response (**genP**).

3.2.1 General Message Request

3.2.1.1 Policy certificate request

genM:: client ----------->> CA
{

pvno	1
sender	User name
recipient	CA name
messageTime	Current time
protectionAlg	**MSG_SIG_ALG** or **MSG_MAC_ALG**
senderKID	Must be present for verification of message protection
transactionID	Implementation-specific (meaningful to client)
senderNonce	128-bit pseudo-random number
freeText	Any text
body	**GenMsgContent**
protection	bits calculated using **MSG_SIG_ALG** or **MSG_MAC_ALG**

} signature (signed using client signing key) or **MAC (key based on authorization code is used to create MAC for structure)**

GenMsgContent ::= SEQUENCE OF InfoTypeAndValue

InfoTypeAndValue ::= SEQUENCE {
infoType OBJECT IDENTIFIER,
infoValue ANY DEFINED BY infoType OPTIONAL }

InfoType ::= private-clientInfo

InfoValue ::= Policy Certificate

3.2.1.2 CA protocol encryption certificate request

genM:: client ----------->> CA
{

pvno	1
sender	User name
recipient	CA name
messageTime	Current time
protectionAlg	**MSG_SIG_ALG** or **MSG_MAC_ALG**
senderKID	Must be present for verification of message protection
transactionID	Implementation-specific (meaningful to client)
senderNonce	128-bit pseudo-random number
freeText	Any text
body	**GenMsgContent**
protection	bits calculated using **MSG_SIG_ALG** or **MSG_MAC_ALG**

} signature (signed using client signing key) or **MAC (key based on authorization code is used to create MAC for structure)**

GenMsgContent ::= SEQUENCE OF InfoTypeAndValue

InfoTypeAndValue ::= SEQUENCE {

infoType OBJECT IDENTIFIER,
infoValue ANY DEFINED BY infoType OPTIONAL }

InfoType ::= CAProtEncCert {id-it 1}

InfoValue ::= Certificate

3.2.2 General Message Response

3.2.2.1 Policy certificate response

genP:: CA ----------->> client
{

pvno	1
sender	CA name
recipient	User name
messageTime	Current time
protectionAlg	**MSG_SIG_ALG** or **MSG_MAC_ALG**
recipKID	Must be present for verification of message protection
transactionID	Implementation-specific (meaningful to client)
senderNonce	Random number from corresponding **genM** message
recipNonce	128-bit pseudo-random number
body	**GenRepContent**
protection	bits calculated using **MSG_SIG_ALG** or **MSG_MAC_ALG**

} signature (signed using CA protocol signing key) or **MAC (key based on authorization code is used to create MAC for structure)**

GenRepContent ::= SEQUENCE OF InfoTypeAndValue

InfoTypeAndValue ::= SEQUENCE {
infoType OBJECT IDENTIFIER,
infoValue ANY DEFINED BY infoType OPTIONAL }

InfoType ::= private-clientInfo

InfoValue ::= Policy Certificate

3.2.2.2 CA protocol encryption certificate response

genP:: CA ----------->> client
{

pvno	1
sender	CA name
recipient	User name
messageTime	Current time
protectionAlg	**MSG_SIG_ALG** or **MSG_MAC_ALG**
recipKID	Must be present for verification of message protection
transactionID	Implementation-specific (meaningful to client)
senderNonce	Random number from corresponding **genM** message
recipNonce	128-bit pseudo-random number
body	**GenRepContent**
protection	bits calculated using **MSG_SIG_ALG** or **MSG_MAC_ALG**

} signature (signed using CA protocol signing key) or **MAC (key based on authorization code is used to create MAC for structure)**

GenRepContent ::= SEQUENCE OF InfoTypeAndValue

```
InfoTypeAndValue ::= SEQUENCE {
        infoType        OBJECT IDENTIFIER,
        infoValue       ANY DEFINED BY infoType  OPTIONAL  }

InfoType ::= CAProtEncCert  {id-it 1}

InfoValue ::= Certificate
```

3.3 Initialization (initialization/registration/certification)

This is the process by which a client first makes itself known to the CA. A successful end result of this operation is the issuance of at least one certificate and the secure initialization of the client's secure storage with its key(s) and certificate(s) along with the CA's public key.

The Initialization exchange consists of three messages:

- Initialization Request (**initReq**),
- Initialization Response (**initRep**), and
- Initialization Confirmation (**pkiConfirm**).

Requirements of a first time initialization are that the client receive out-of-band information from the CA or RA. That information being a reference number and authentication code. The client may initialize in the following ways:

- Request for verification certificate only (client always supplies key pair), or
- Client requests both a verification certificate and an encryption certificate. Either the CA or the client can provide encryption key pair.

3.3.1 Verification Request Only

Client must always provide a public key and prove possession of the private key in the request message.

3.3.2 Verification and Encryption Request

The way in which this type of request takes place varies depending on where the encryption keys originate, client or at the CA, and, when at the client, whether or not the private component is sent for archive and/or proof of possession.

The variants can be broken down into the following cases:

- Encryption key pair originates at the CA,
- Encryption key pair originates at client but sends private key, and
- Encryption key pair originates at client but client does not send private key.

3.3.2.1 Encryption key pair originates at the CA

- the client will include an empty public key field in the **CertTemplate** of the encryption portion of the request from the client; and
- the client will generate a protocol encryption key pair and send the public component in the request message.

3.3.2.2 Encryption key pair originates at the client

A public key value in the **CertTemplate** of the encryption portion of the request indicates the client is providing the encryption key pair.

3.3.2.2.1 Private key sent

If the client sends the private component of the encryption key pair for either archival or proof of possession purposes the client must in turn first retrieve the CA's protocol encryption certificate using a General Message (**genM**) exchange (refer to Section 3.2.1.2). The client will use the public key in the protocol encryption certificate to protect a symmetric key which in turn is used to protect the private key information sent in the message.

3.3.2.2.2 Private key not sent

If the client does not send the private component of the encryption key pair the message flow changes in the following way.

Within the certificate response, the newly generated encryption certificate must be encrypted by the CA with a fresh symmetric key (CA's symmetric key) and this symmetric key must be encrypted with the public key that appears in the newly created encryption certificate. This is necessary to establish proof of possession. This is achieved by the client demonstrating it's ability to decrypt the certificate. The final confirmation from the client must be protected with a newly generated symmetric key based on the bits of CA's symmetric key received in the response.

3.3.3 Message Authentication

Message origin authentication is provided by a password-based MAC by both the client and the CA.

3.3.4 Message Flow Behavior

The client will always initiate the exchange with a certificate request message.

The CA will provide a response that contains new certificate(s) and the CA certificate upon success. If the CA rejects the request, an error message must be returned detailing the reason for failure. Human-readable reason failure text must be returned.

The client may accept this response from the CA and send a confirmation message, or may reject the certificate(s) with either an error message detailing why the rejection took place or no confirmation at all. Note that a rejection implies all certificates in the response have been rejected as the protocol does not provide any mechanism by which selective rejection may occur.

The CA should revoke the newly created certificate(s) if anything other than a confirmation is received from the client.

3.3.5 Message Flows

3.3.5.1 Initialization request

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

initReq:: client ----------->> CA
{

pvno	1
sender	User name
recipient	CA name
messageTime	Current time
protectionAlg	**MSG_MAC_ALG**
senderKID	Reference number
transactionID	Implementation-specific (meaningful to client)
senderNonce	128-bit pseudo-random number
freeText	Text
body	**CertReqMessages**
protection	bits calculated using **MSG_MAC_ALG**

} MAC (key based on authorization code is used to create MAC for structure)

The message is verified by the CA. If valid, the CA then creates a new certificate for the client. The CA responds with an **initRep** message to the client.

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

CertReqMsg ::= SEQUENCE {
certReq CertRequest,
pop ProofOfPossession OPTIONAL,
-- content depends upon key type **}**

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

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

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

CertTemplate ::= SEQUENCE {

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

The extensions recognized by the CA that can be included in a PKIX-CMP initialization request are: **authorityKeyIdentifier**, **subjectKeyIdentifier**, **keyUsage**, **extKeyUsage**,

privateKeyUsagePeriod, **certificatePolicies**, **privateVersInfo**, **subjectAltName**, **basicConstraints**, and **cRLDistributionPoints**.

```
AlgorithmIdentifier  ::=  SEQUENCE  {
        Algorithm               OBJECT IDENTIFIER,
        Parameters              ANY DEFINED BY algorithm OPTIONAL  }

SubjectPublicKeyInfo  ::=  SEQUENCE  {
        Algorithm               AlgorithmIdentifier,
        SubjectPublicKey        BIT STRING  }

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

AttributeTypeAndValue ::= SEQUENCE {

        type    OBJECT IDENTIFIER,
        value   ANY DEFINED BY type }

id-regCtrl-regToken             OBJECT IDENTIFIER ::= { id-regCtrl 1 }
id-regCtrl-authenticator        OBJECT IDENTIFIER ::= { id-regCtrl 2 }
id-regCtrl-pkiPublicationInfo   OBJECT IDENTIFIER ::= { id-regCtrl 3 }
id-regCtrl-pkiArchiveOptions    OBJECT IDENTIFIER ::= { id-regCtrl 4 }
id-regCtrl-oldCertID            OBJECT IDENTIFIER ::= { id-regCtrl 5 }
id-regCtrl-protocolEncrKey      OBJECT IDENTIFIER ::= { id-regCtrl 6 }

ProofOfPossession ::= POPOSigningKey
```

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. 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 is demonstrated by the inclusion of the private key (encrypted) in the **CertRequest** (in the **PKIArchiveOptions** control structure).

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

AlgorithmIdentifier  ::=  SEQUENCE  {

        Algorithm               OBJECT IDENTIFIER,
        Parameters              ANY DEFINED BY algorithm OPTIONAL  }

POPOSigningKeyInput ::= SEQUENCE {
        publicKeyMAC            PKMACValue
        publicKey               SubjectPublicKeyInfo }

SubjectPublicKeyInfo  ::=  SEQUENCE  {
        Algorithm               AlgorithmIdentifier,
        SubjectPublicKey        BIT STRING  }

AlgorithmIdentifier  ::=  SEQUENCE  {
        Algorithm               OBJECT IDENTIFIER,
        Parameters              ANY DEFINED BY algorithm OPTIONAL  }

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

```
        value    BIT STRING }

PBMParameter ::= SEQUENCE {
        salt                    OCTET STRING,
        owf                     AlgorithmIdentifier,
        -- AlgId for a One-Way Function (SHA-1 recommended)
        iterationCount          INTEGER,
        -- number of times the OWF is applied
        mac                     AlgorithmIdentifier
        -- the MAC AlgId  }
```

3.3.5.2 Initialization response

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

initRep:: CA ----------->> client

{

pvno	1
sender	CA name
recipient	User name
messageTime	Time at which CA produced message
protectionAlg	**MSG_MAC_ALG**
recipKID	Reference number
transactionID	Value from corresponding **initReq** message
senderNonce	Value from corresponding **initReq** message
recipNonce	128-bit pseudo-random number
freeText	Text
body	**CertRepMessage**
certificate	Present
privateKey	Present
protection	bits calculated using **MSG_MAC_ALG**

} MAC (key based on authorization code is used to create MAC for structure)

The message is verified by the client. It then sends a **pKIConfirm** message to the CA.

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

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

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

CertifiedKeyPair ::= SEQUENCE {
```

certOrEncCert **CertOrEncCert,**
privateKey **[0] EncryptedValue OPTIONAL,**
publicationInfo **[1] PKIPublicationInfo OPTIONAL }**

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

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

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

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

3.3.5.3 Initialization confirmation

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

pkiConfirm:: client ----------->> CA
{

pvno	1
sender	User name
recipient	CA name
messageTime	Current time
protectionAlg	**MSG_MAC_ALG**
senderKID	Reference number
transactionID	Value from corresponding **initReq** message
senderNonce	Value from corresponding **initReq** message
recipNonce	Value from corresponding **initRep** message
freeText	Text
protection	bits calculated using **MSG_MAC_ALG**

} MAC (key based on authorization code is used to create MAC for structure)

3.4 Certificate Request

This is the process by which a client requests additional certificates. For example, if in the first time initialization only a verification certificate was issued, the client may return to the CA for an encryption certificate. This operation requires that the client already be initialized with valid keys capable of signature operations. This is necessary for message authentication.

This section provides a profile of the PKIX-CMP certificate request message and the corresponding response and confirmation messages for PKI enrollment of subscribers. This section indicates which fields must be present in the request message and how their contents are derived. This section also indicates what responses will be returned under various circumstances and how the requester shall behave on receipt of those responses.

The Certificate Request exchange consists of three messages:

- Certificate Request (**certReq**),
- Certificate Response (**certRep**), and

- Certificate Confirmation (**pkiConfirm**).

3.4.1 Message Authentication

Message protection is provided by digital signature.

The client must be in possession of at least one valid signing key in order to authenticate each message sent to the CA and must be in possession of the current CA certificate.

The CA will use a protocol signing key to protect certificate request messages. The protocol verification certificate must be included in any response from the CA in which the protocol signing key is used for message authentication.

3.4.2 Message Flow Behavior

Message flow also varies as described when encryption certificate requests are done. This variance again depends on where the encryption key pair originates, client or CA, and whether or not the private component is sent to the CA in the former case.

A client may request any additional certificates through this message. Up to two certificates may be updated in a single request. For each certificate to be updated, the client must provide the serial number of the said certificate. Failure to do so should result in an error message from the CA.

Any new certificate request must include key usage settings which differentiate the certificate from those valid certificates which the client already possesses. Failure to do so should result in an error message from the CA. That is, if the client already possesses a certificate with a key usage of **digitalSignature**, then a request for another certificate with **digitalSignature** key usage will result in an error from the CA. On the other hand, if the certificate request contains a different key usage (e.g., **nonRepudiation**) the request should be granted. A key usage combination such as **digitalSignature** with **nonRepudiation** is different from a key usage of just **digitalSignature**.

The addition of this message opens up the number of keys a client may possess, which is great for future growth. Currently, up to three key pairs (key usage in brackets) is supported:

- Verification (**digitalSignature** - this includes dual-purpose **digitalSignature** and **keyEncipherment**)
- Non-repudiation (**nonRepudiation**)
- Encryption (**keyEncipherment**)

3.4.3 Message Flows

3.4.3.1 Certificate request

certReq:: client ----------->> CA
{

pvno	1
sender	User name
recipient	CA name
messageTime	Current time
protectionAlg	**MSG_SIG_ALG** (any authenticated protection algorithm)
senderKID	For verification of message protection
transactionID	Implementation-specific (meaningful to client)
senderNonce	128-bit pseudo-random number
freeText	Text
body	**CertReqMessages**

protection bits calculated using **MSG_SIG_ALG**
} Signature (signed with client signing key)

The CA will use a protocol signing key to protect certificate request messages. The protocol verification certificate must be included in any response from the CA in which the protocol signing key is used for message authentication.

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

Alternatively, the **PKIBody** may be a **CertificationRequest** (this structure is fully specified by the ASN.1 structure **CertificationRequest** given in Section 7, Reference 4). This structure may be required for certificate requests for signing key pairs when interoperation with legacy systems is desired, but its use is strongly discouraged whenever not absolutely necessary

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

CertReqMsg ::= SEQUENCE {
certReq CertRequest,
pop ProofOfPossession OPTIONAL,
-- content depends upon key type **}**

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

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

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

CertTemplate ::= SEQUENCE {

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

The extensions recognized by the CA that can be included in a PKIX-CMP certificate request are: **authorityKeyIdentifier**, **subjectKeyIdentifier**, **keyUsage**, **extKeyUsage**, **privateKeyUsagePeriod**, **certificatePolicies**, **privateVersInfo**, **subjectAltName**, **basicConstraints**, and **cRLDistributionPoints**.

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

AttributeTypeAndValue ::= SEQUENCE {
        type     OBJECT IDENTIFIER,
        value    ANY DEFINED BY type }

id-regCtrl-regToken             OBJECT IDENTIFIER ::= { id-regCtrl 1 }
id-regCtrl-authenticator        OBJECT IDENTIFIER ::= { id-regCtrl 2 }
id-regCtrl-pkiPublicationInfo   OBJECT IDENTIFIER ::= { id-regCtrl 3 }
id-regCtrl-pkiArchiveOptions    OBJECT IDENTIFIER ::= { id-regCtrl 4 }
id-regCtrl-oldCertID            OBJECT IDENTIFIER ::= { id-regCtrl 5 }
id-regCtrl-protocolEncrKey      OBJECT IDENTIFIER ::= { id-regCtrl 6 }

ProofOfPossession ::= CHOICE {
        signature                       [1] POPOSigningKey,
        keyEncipherment         [2] POPOPrivKey }
```

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. 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 is demonstrated by the inclusion of the private key (encrypted) in the **CertRequest** (in the **PKIArchiveOptions** control structure).

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

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

        Algorithm               OBJECT IDENTIFIER,
        Parameters              ANY DEFINED BY algorithm OPTIONAL  }

SubjectPublicKeyInfo  ::=  SEQUENCE  {
        Algorithm               AlgorithmIdentifier,
        SubjectPublicKey        BIT STRING  }

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

PBMParameter ::= SEQUENCE {
        salt            OCTET STRING,
        owf              AlgorithmIdentifier,
```

-- AlgId for a One-Way Function (SHA-1 recommended)
iterationCount INTEGER,
-- number of times the OWF is applied
mac AlgorithmIdentifier
-- the MAC AlgId **}**

3.4.3.2 Certificate response

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

certRep:: CA ----------->> client
{

pvno	1
sender	CA name
recipient	User name
messageTime	Current time
protectionAlg	**MSG_SIG_ALG**
recipKID	For verification of message protection
transactionID	Value from corresponding **certReq** message
senderNonce	Value from corresponding **certReq** message
recipNonce	128-bit pseudo-random number
freeText	Text
body	**CertRepMessages**
certificate	Present
privateKey	Present
protection	bits calculated using **MSG_SIG_ALG**
extraCerts	CA's protocol verification certificate

} Signature (signed with CA protocol signing key)

The message is verified by the client. It then sends a **pKIConfirm** message to the CA.

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

CertResponse ::= SEQUENCE {

certReqId INTEGER,
-- to match this response with corresponding request (a value
-- of -1 is to be used if ***certReqId*** *is not specified in the*
-- corresponding request)
status PKIStatusInfo,
certifiedKeyPair CertifiedKeyPair OPTIONAL,
rspInfo OCTET STRING OPTIONAL }

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

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

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

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

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

3.4.3.3 Certificate request confirmation

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

pkiConfirm:: client ---------->> CA
{

pvno	1
sender	User name
recipient	CA name
messageTime	Current time
transactionID	Value from corresponding **certReq** message
senderNonce	Value from corresponding **certReq** message
recipNonce	Value from corresponding **certRep** message
protectionAlg	**MSG_SIG_ALG**
senderKID	For verification of message protection
freeText	Text
protection	bits calculated using **MSG_SIG_ALG**

} Signature (signed with client signing key)

3.5 Key Update

This section provides a profile of the PKIX-CMP renewal/update message and the corresponding response message for PKI certificate renewal by subscribers and subject CAs. It indicates which fields must be present in the request message and how their contents are derived. It also indicates what responses will be returned under various circumstances and how the requester shall behave on receipt of those responses.

For updating both key pairs, the client must identify within the request the certificates to be updated by supplying the certificate serial numbers.

The client always supplies the public key and must prove possession of the private component. The client must also provide the serial number of the verification certificate which is being updated.

The client performs the following steps in a key update request. First, the client generates a random number. Then either the security application provides the client engine with the signing key pair and the certificate templates that it wants to obtain certificate(s) or the security application requests the client engine to create a signature key pair. The client

then sends a **keyUpdReq** message to the CA. This message is authenticated using the client's signing private key.

The Key Update exchange consists of three messages:

- Key Udpate Request (**keyUpdReq**),
- Key Udpate Response (**keyUpdRep**), and
- Key Udpate Confirmation (**pkiConfirm**).

3.5.1 Message Authentication

Message protection differs from the first time initialization exchange for both the client and the CA.

The client must be in possession of at least one valid signing key in order to authenticate each message sent to the CA and must be in possession of the current CA certificate.

The CA will use a protocol signing key to protect update request messages. The protocol verification certificate must be included in any response from the CA in which the protocol signing key is used for message authentication.

3.5.2 Message Flow Behavior

Message flow varies as described when encryption certificate updates are done. This variance again depends on where the encryption key pair originates, client or CA, and whether or not the private component is sent to the CA in the former case.

Any new certificate request must include key usage settings that differentiate the certificate from those valid certificates which the client already possesses. Failure to do so should result in an error message from the CA. That is, if the client already possesses a certificate with a key usage of **digitalSignature**, then a request for another certificate with **digitalSignature** key usage will result in an error from the CA. On the other hand, if the certificate request contains a differing key usage (such as non-repudiation) the request should be granted assuming other criteria are met in the request message.

3.5.2.1 Single Certificate Update

3.5.2.1.1 Verification certificate update

As in the first time initialization, client always supplies the public key and must prove possession of the private component. The client must also provide the serial number of the verification certificate that is being updated.

3.5.2.1.2 Encryption certificate update

Again, as in the first time initialization a general message exchange may be necessary if the client is to send the private component of the encryption key pair. Furthermore, the client must send a protocol encryption key if the CA is providing the encryption key pair. Proof of possession must be established as always and may result in this being accomplished in the confirmation message from the client.

3.5.2.2 Two Certificate Update

In this case the CA is being asked to update two certificates which would typically be a verification certificate and an encryption certificate. The client must identify within the request the certificates to be updated by supplying the certificate serial numbers.

3.5.3 Message Flows

3.5.3.1 Key update request

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

keyUpdReq:: client ----------->> CA
{

pvno	1
sender	User name
recipient	CA name
messageTime	Current time
protectionAlg	**MSG_SIG_ALG**
senderKID	For verification of message protection
transactionID	Implementation-specific (meaningful to client)
senderNonce	128-bit pseudo-random number
freeText	Text
body	**CertReqMessages**
protection	bits calculated using **MSG_SIG_ALG**

} Signature (signed with client signing key)

The CA will use a protocol signing key to protect update request messages. The protocol verification certificate must be included in any response from the CA in which the protocol signing key is used for message authentication.

The message is verified by the CA. If valid, the CA will create a verification certificate for the client and respond with a **keyUpdRep** message.

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

CertReqMsg ::= SEQUENCE {
certReq **CertRequest,**
pop **ProofOfPossession OPTIONAL,**
-- content depends upon key type **}**

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

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

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

CertTemplate ::= SEQUENCE {
version **[0] Version** **OPTIONAL,**
serialNumber **[1] INTEGER** **OPTIONAL,**

```
signingAlg      [2] AlgorithmIdentifier          OPTIONAL,
issuer          [3] Name                 OPTIONAL,
validity        [4] OptionalValidity             OPTIONAL,
subject         [5] Name                 OPTIONAL,
publicKey       [6] SubjectPublicKeyInfo OPTIONAL,
issuerUID       [7] UniqueIdentifier             OPTIONAL,
subjectUID      [8] UniqueIdentifier             OPTIONAL,
extensions      [9] Extensions           OPTIONAL }
```

The extensions recognized by the CA that can be included in a PKIX-CMP key update request are: **authorityKeyIdentifier**, **subjectKeyIdentifier**, **keyUsage**, **extKeyUsage**, **privateKeyUsagePeriod**, **certificatePolicies**, **privateVersInfo**, **subjectAltName**, **basicConstraints**, and **cRLDistributionPoints**.

```
AlgorithmIdentifier ::= SEQUENCE {
        Algorithm               OBJECT IDENTIFIER,
        Parameters              ANY DEFINED BY algorithm OPTIONAL  }

SubjectPublicKeyInfo ::= SEQUENCE {
        Algorithm               AlgorithmIdentifier,
        SubjectPublicKey        BIT STRING  }

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

AttributeTypeAndValue ::= SEQUENCE {
        type    OBJECT IDENTIFIER,
        value   ANY DEFINED BY type }

id-regCtrl-regToken             OBJECT IDENTIFIER ::= { id-regCtrl 1 }
id-regCtrl-authenticator        OBJECT IDENTIFIER ::= { id-regCtrl 2 }
id-regCtrl-pkiPublicationInfo   OBJECT IDENTIFIER ::= { id-regCtrl 3 }
id-regCtrl-pkiArchiveOptions    OBJECT IDENTIFIER ::= { id-regCtrl 4 }
id-regCtrl-oldCertID            OBJECT IDENTIFIER ::= { id-regCtrl 5 }
id-regCtrl-protocolEncrKey      OBJECT IDENTIFIER ::= { id-regCtrl 6 }

ProofOfPossession ::= POPOSigningKey
```

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. 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 is demonstrated by the inclusion of the private key (encrypted) in the **CertRequest** (in the **PKIArchiveOptions** control structure).

```
POPOSigningKey ::= SEQUENCE {
        algorithmIdentifier     AlgorithmIdentifier,
        signature               BIT STRING }
```

3.5.3.2 Key update response

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

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

keyUpdRep:: CA ----------->> client
{

pvno	1
sender	CA name
recipient	User name
messageTime	Current time
protectionAlg	**MSG_SIG_ALG**
recipKID	For verification of message protection
transactionID	Value from corresponding **keyUpdReq** message
senderNonce	Value from corresponding **keyUpdReq** message
recipNonce	128-bit pseudo-random number
freeText	Text
body	**CertRepMessage**
certificate	Present
privateKey	Present
protection	bits calculated using **MSG_SIG_ALG**
extraCerts	Present

} Signature (signed with CA protocol signing key)

The message is verified by the client. It then sends a **pKIConfirm** message to the CA.

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

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

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


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

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

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

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

3.5.3.3 Key update confirmation

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

pkiConfirm:: client ----------->> CA
{

pvno	1
sender	CA name
recipient	User name
messageTime	Current time
transactionID	Value from corresponding **keyUpdReq** message
senderNonce	Value from corresponding **keyUpdRep** message
recipNonce	Value from corresponding **keyUpdRep** message
protectionAlg	**MSG_SIG_ALG**
senderKID	For verification of message protection
freeText	Text
protection	bits calculated using **MSG_SIG_ALG**

} Signature (signed with client signing key)

3.6 Cross-certification

This section provides a profile of the PKIX-CMP request message and the corresponding response and confirmation messages for PKI enrollment of subject CAs. Indicates which fields must be present in the request message and how their contents are derived. Also indicates what responses will be returned under various circumstances and how the requester shall behave on receipt of those responses.

The message flow is the same as all requests that result in the generation of a certificate. Hence, a confirmation message from the client is required.

The Cross-certification exchange consists of three messages:

- Cross-certification Request (**certReq**),
- Cross-certification Response (**certRep**), and
- Cross-certification Confirmation (**pkiConfirm**).

3.6.1 Message Flows

3.6.1.1 Cross-certification request

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.

certReq:: CA1 ----------->> CA2
{

pvno	1
sender	Name of CA1
recipient	Name of CA2
messageTime	Time of production of message
protectionAlg	**MSG_MAC_ALG**

senderKID	Reference number
transactionID	Implementation-specific (meaningful to CA)
senderNonce	128-bit pseudo-random number
freeText	Text
body	**CertReqMessage**
protection	bits calculated using **MSG_MAC_ALG**

} MAC (key based on authorization code is used to create MAC for structure)

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

CertReqMsg ::= SEQUENCE {
certReq **CertRequest,**
pop **ProofOfPossession OPTIONAL**
-- content depends upon key type **}**

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

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

CertRequest ::= SEQUENCE {
certReqId **INTEGER,**
-- ID for matching request and reply
certTemplate **CertTemplate**
-- *Selected fields of cert to be issued* **}**

CertTemplate ::= SEQUENCE {
version **[0] Version** **OPTIONAL,**
-- v1 (0) or v3 (2)
signingAlg **[2] AlgorithmIdentifier** **OPTIONAL,**
-- Requesting CA must know in advance with which algorithm it wishes the certificate -- to be signed
issuer **[3] Name** **OPTIONAL,**
-- may be null only if ***issuerAltName*** *extension value proposed*
validity **[4] OptionalValidity** **OPTIONAL,**
subject **[5] Name** **OPTIONAL,**
-- may be null only if ***subjectAltName*** *extension value proposed*
publicKey **[6] SubjectPublicKeyInfo** **OPTIONAL,**
-- the CA key to be certified
extensions **[9] Extensions** **OPTIONAL**
-- requesting CA must propose values for all extensions which it requires to be in the -- cross-certificate **}**

The extensions recognized by the CA that can be included in a PKIX-CMP cross-certification request are: **authorityKeyIdentifier**, **subjectKeyIdentifier**, **keyUsage**, **extKeyUsage**, **privateKeyUsagePeriod**, **certificatePolicies**, **privateVersInfo**, **subjectAltName**, **basicConstraints**, and **cRLDistributionPoints**.

AlgorithmIdentifier ::= SEQUENCE {
Algorithm **OBJECT IDENTIFIER,**
Parameters **ANY DEFINED BY algorithm OPTIONAL }**

SubjectPublicKeyInfo ::= SEQUENCE {
Algorithm **AlgorithmIdentifier,**

SubjectPublicKey BIT STRING }

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

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

ProofOfPossession ::= POPOSigningKey

Since the cross-certification request is a request for a verification certificate, the proof of possession of the CA signing key is demonstrated through use of the **POPOSigningKey** structure.

POPOSigningKey ::= SEQUENCE {
algorithmIdentifier AlgorithmIdentifier,
signature BIT STRING }

The CA2 will use its signing key to protect cross-certificate request messages. The CA1 verification certificate must be included in any response from the CA2 in which the protocol signing key is used for message authentication.

3.6.1.2 Cross-certification response

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

certRep:: CA2 ----------->> CA1
{

pvno	1
sender	Name of CA2
recipient	Name of CA1
messageTime	Time at which CA produced message
protectionAlg	**MSG_MAC_ALG**
senderKID	For verification of message protection
recipKID	Reference number
transactionID	Value from corresponding **certReq** message
senderNonce	Value from corresponding **certReq** message
recipNonce	128-bit pseudo-random number
freeText	Text
body	**CertRepMessage**
response	present
status	present - PKI status indicator
failInfo	present depending on status
certifiedKeyPair	Certified key pair
certificate	Cross-certificate
protection	bits calculated using **MSG_MAC_ALG**
extraCerts	CA's verification certificate

} MAC (key based on authorization code is used to create MAC for structure)

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

CertResponse ::= SEQUENCE {

```
        certReqId                   INTEGER,
        -- to match this response with corresponding request (a value
        -- of -1 is to be used if certReqId is not specified in the
        -- corresponding request)
        status                  PKIStatusInfo,
        certifiedKeyPair        CertifiedKeyPair        OPTIONAL,
        rspInfo                     OCTET STRING        OPTIONAL }

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

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

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

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

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

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

The requesting CA1 then sends a **pKIConfirm** message to the recipient CA2.

3.6.1.3 Cross-certification confirmation

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

pkiConfirm:: CA1 ----------->> CA2
{

pvno	1
sender	Name of CA1
recipient	Name of CA2
transactionID	Value from corresponding **certReq** message
senderNonce	Value from corresponding **certRep** message
recipNonce	Value from corresponding **certRep** message
protectionAlg	**MSG_MAC_ALG**
senderKID	Reference number
freeText	Text
protection	bits calculated using **MSG_MAC_ALG**

} MAC (key based on authorization code is used to create MAC for structure)

3.7 Key Recovery

This section provides a profile of the PKIX-CMP request message and the corresponding response and confirmation messages for PKI key recovery of subscribers. It indicates which fields must be present in the request message and how their contents are derived. It also indicates what responses will be returned under various circumstances and how the requester shall behave on receipt of those responses.

The client receives the authorization code and a reference number for use in identifying the client and the CA and for authenticating the exchanged messages. The engine sends a **keyRecReq** message to the CA. This message is secured using a MAC derived from the authorization code.

This exchange is performed when a client no longer has a valid signature key pair or client key materials have been lost. It is used to obtain the encryption key history from the CA.

The Key Recovery request exchange consists of three messages:

- Key Recovery Request (**keyRecReq**),
- Key Recovery Response (**keyRecRep**), and
- Key Recovery Confirmation (**pkiConfirm**).

3.7.1 Message Authentication

Message will be protected with a password-based MAC. The client must obtain the reference number and authentication code out-of-band.

3.7.2 Message Flow Behavior

A client requesting key recovery will always request a new verification certificate as well as the encryption key history.

It is not necessary to include key usage within each of the certificate requests.

3.7.3 Message Flows

3.7.3.1 Key recovery request

keyRecReq:: client ----------->> CA
{

pvno	1
sender	User name
recipient	CA name
messageTime	Current time
protectionAlg	**MSG_MAC_ALG**
senderKID	Reference number
transactionID	Implementation-specific (meaningful to client)
senderNonce	128-bit pseudo-random number
freeText	Text
body	**CertReqMessages**
protection	bits calculated using **MSG_MAC_ALG**

} MAC (key based on authorization code is used to create MAC for structure)

For key recovery requests the syntax used is identical to the initialization request **CertReqMessages**. Typically, **SubjectPublicKeyInfo** and **KeyId** are the template fields which may be used to supply a signature public key for which a certificate is required.

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

```
CertReqMsg ::= SEQUENCE {
        certReq                 CertRequest,
        pop                     ProofOfPossession  OPTIONAL
        -- content depends upon key type }
```

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

Note that if a key history is required, the requester must supply a Protocol Encryption Key control in the request message.

The message is verified by the CA. If valid, the CA then creates a verification certificate for the client and prepares decryption key history and encryption certificates to send to client. The CA responds with a **keyRecRep** message to the client.

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

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

The extensions recognized by the CA that can be included in a PKIX-CMP cross-certification request are: **authorityKeyIdentifier**, **subjectKeyIdentifier**, **keyUsage**, **extKeyUsage**, **privateKeyUsagePeriod**, **certificatePolicies**, **privateVersInfo**, **subjectAltName**, **basicConstraints**, and **cRLDistributionPoints**.

```
AlgorithmIdentifier  ::=  SEQUENCE  {
        Algorithm               OBJECT IDENTIFIER,
        Parameters              ANY DEFINED BY algorithm OPTIONAL  }

SubjectPublicKeyInfo  ::=  SEQUENCE  {
        Algorithm               AlgorithmIdentifier,
        SubjectPublicKey        BIT STRING  }

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

AttributeTypeAndValue ::= SEQUENCE {
        type    OBJECT IDENTIFIER,
        value   ANY DEFINED BY type }

id-regCtrl-regToken             OBJECT IDENTIFIER ::= { id-regCtrl 1 }
id-regCtrl-authenticator        OBJECT IDENTIFIER ::= { id-regCtrl 2 }
id-regCtrl-pkiPublicationInfo   OBJECT IDENTIFIER ::= { id-regCtrl 3 }
id-regCtrl-pkiArchiveOptions    OBJECT IDENTIFIER ::= { id-regCtrl 4 }
```

id-regCtrl-oldCertID **OBJECT IDENTIFIER ::= { id-regCtrl 5 }**
id-regCtrl-protocolEncrKey **OBJECT IDENTIFIER ::= { id-regCtrl 6 }**

3.7.3.2 Key recovery response

keyRecRep:: CA ----------->> client
{

pvno	1
sender	CA name
recipient	User name
messageTime	Current time
protectionAlg	**MSG_MAC_ALG**
recipKID	Reference number
transactionID	Value from corresponding **keyRecReq** message
senderNonce	Value from corresponding **keyRecReq** message
recipNonce	128-bit pseudo-random number
freeText	Text
body	**KeyRecRepContent**
certificate	Present
privateKey	Present
protection	bits calculated using **MSG_MAC_ALG**

} MAC (key based on authorization code is used to create MAC for structure)

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 SIZE (1..MAX) OF Certificate OPTIONAL,
        keyPairHist     [2] SEQUENCE SIZE (1..MAX) OF CertifiedKeyPair OPTIONAL
}

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

The message is verified by the client. The verification certificate and encryption key history are given to the security application. It then sends a **pKIConfirm** message to the CA.

3.7.3.3 Key recovery confirmation

pkiConfirm:: client ----------->> CA
{

pvno	1
sender	User name
recipient	CA name
transactionID	Value from corresponding **keyRecRep** message
senderNonce	Value from corresponding **keyRecRep** message
recipNonce	Value from corresponding **keyRecRep** message
protectionAlg	**MSG_MAC_ALG**
senderKID	Reference number
freeText	Text
protection	bits calculated using **MSG_MAC_ALG**

} MAC (key based on authorization code is used to create MAC for structure)

3.8 Revocation

This section provides a profile of the PKIX-CMP revocation request message and the corresponding response message for PKI certificate revocation by subscribers and subject CAs. It indicates which fields must be present in the request message and how their contents are derived. It also indicates what responses will be returned under various circumstances and how the requester shall behave on receipt of those responses.

This exchange is performed when a client needs to revoke his or her own certificates. This transaction requires that clients be in possession of at least one valid signing key.

A client can request revocation of many certificates in a single message. The CA will respond with a status for each of the certificates that were requested to be revoked. There is no confirmation message from the client.

For each certificate to be revoked, the client must provide a means for the CA to determine the target certificate (e.g., certificate serial number).

The Key Revocation request exchange consists of two messages:

- Revocation Request (**revReq**) and
- Revocation Response (**revRep**).

3.8.1 Message Authentication

Client authenticates message with a valid signing key. CA authenticates message with a protocol signing key.

3.8.2 Message Flow Behavior

A client can request revocation of many certificates in a single message. The CA will respond with a status for each of the certificates that were requested to be revoked. There is no confirmation message from the client as it is not necessary in this case.

For each certificate to be revoked, the client must provide a means for the CA to determine the target certificate. This will be accomplished by either:

- The client provides the actual serial number of the certificate to be revoked, or
- The client provides Distinguished Name and the exact key usage of the certificate to be revoked.

Optionally, the client may specify a revocation reason (**ReasonFlags**), a **GeneralizedTime** representing a not valid since date as well as **crlEntryExtensions**.

The Reason code **certificateHold** is not supported. Any requests containing this reason code will be rejected by the CA.

The CA must ensure that the client requesting the revocation is actually the subject of the revocation for each request within the message. This restriction is lifted for an RA client. The CA will respond with a message detailing the status of each of the revocation requests.

3.8.3 Message Flows

3.8.3.1 Revocation request

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.

revReq:: client ----------->> CA
{

pvno	1
sender	User name
protectionAlg	**MSG_SIG_ALG**
senderKID	For verification of message protection
transactionID	Implementation-specific (meaningful to client)
senderNonce	128-bit pseudo-random number
freeText	Text
body	**RevReqContent**
protection	bits calculated using **MSG_SIG_ALG**

} Signature (signed with client signing key)

RevReqContent ::= SEQUENCE OF RevDetails

RevDetails ::= SEQUENCE {
certDetails CertTemplate,
-- allows requester to specify as much as they can about the cert. for which

-- revocation is requested (e.g., for cases in which serialNumber is not available)
revocationReason ReasonFlags OPTIONAL,
-- the reason that revocation is requested
badSinceDate GeneralizedTime OPTIONAL,
-- indicates best knowledge of sender
crlEntryDetails Extensions OPTIONAL
-- requested crlEntryExtensions **}**

CertTemplate ::= SEQUENCE {

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

AlgorithmIdentifier ::= SEQUENCE {

Algorithm	**OBJECT IDENTIFIER,**
Parameters	**ANY DEFINED BY algorithm OPTIONAL }**

SubjectPublicKeyInfo ::= SEQUENCE {

Algorithm	**AlgorithmIdentifier,**
SubjectPublicKey	**BIT STRING }**

The message is verified by the CA. If valid, the CA revokes the certificate in question for the client and sends a **modifyRequest** message to the directory to issue a new CRL including the revoked certificate. The CA responds with a **revRep** message to the client.

3.8.3.2 Revocation response

The revocation response is sent to the requester of the revocation.

revRep:: CA ----------->> client
{

pvno	1

sender	CA name
recipient	User name
messageTime	Current time
protectionAlg	**MSG_SIG_ALG**
recipKID	Reference number
transactionID	Value from corresponding **revReq** message
senderNonce	Value from corresponding **revReq** message
recipNonce	128-bit pseudo-random number
freeText	Text
body	**RevRepContent**
certificate	Present
privateKey	Present
protection	bits calculated using **MSG_SIG_ALG**
extraCerts	CA's protocol verification certificate

} Signature (signed with CA protocol signing key)

```
RevRepContent ::= SEQUENCE {
        status          SEQUENCE SIZE (1..MAX) OF PKIStatusInfo,
        revCerts        [0] SEQUENCE SIZE (1..MAX) OF CertId OPTIONAL,
        crls            [1] SEQUENCE SIZE (1..MAX) OF CertificateList  OPTIONAL }

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

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

CertificateList  ::=  SEQUENCE  {
      tbsCertList                TBSCertList,
      signatureAlgorithm         AlgorithmIdentifier,
      signatureValue                   BIT STRING  }

TBSCertList  ::=  SEQUENCE  {
        version                          Version OPTIONAL,
        -- if present, shall be v2
        signature                        AlgorithmIdentifier,
        issuer                   Name,
        thisUpdate                       Time,
        nextUpdate                       Time OPTIONAL,
        revokedCertificates      SEQUENCE OF SEQUENCE  {
                userCertificate          CertificateSerialNumber,
                revocationDate           Time,
                crlEntryExtensions       Extensions OPTIONAL
                -- if present, shall be v2 }  OPTIONAL,
        crlExtensions            [0]  EXPLICIT Extensions OPTIONAL
        -- if present, shall be v2 }

AlgorithmIdentifier  ::=  SEQUENCE  {
        Algorithm                OBJECT IDENTIFIER,
        Parameters               ANY DEFINED BY algorithm OPTIONAL  }
```

4 CA Key Update

4.1 Introduction

This section describes an overview of the CA key update procedure and indicates how certificate users obtain the new CA key. This mechanism does not represent a PKIX-CMP message transaction.

4.2 Root CA key update

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

- **OldWithOld**: a true self-signed certificate. The contained public key must be usable to verify the signature (integrity only);
- **OldWithNew**: the previous root CA public key signed with new private key;
- **NewWithOld**: new root CA public key signed with previous private key; and
- **NewWithNew**: new root CA public key signed with new private key.

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 entities' certificates expire.

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

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.

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

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

After a CA key update has occurred, it is important that users update their CA trust anchor to be the new CA verification public key sometime before their old CA trust anchor expires. This is accomplished when the user contacts the CA to perform a key recovery or key update operation. During this operation, the Client is passed its new CA trust anchor, which it securely stores in its profile. After receiving a new CA trust anchor, all client certificate validations must lead from the new CA trust anchor to the certificate being validated.

4.3 CA Operator actions

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

1) Generates a new key pair;

2) Creates a certificate containing the old CA public key signed with the new private key (the "old with new" certificate);

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

4) Creates a certificate containing the new CA public key signed with the new private key (the "new with new" certificate);

5) Publishes these new certificates via the directory; and

6) Exports the new CA public key so that end entities may acquire it using the "out-of-band" mechanism (if required).

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

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

The "new with old" certificate 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 (at the latest, the expiry date of the old 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 by which the CA will next update its key pair.

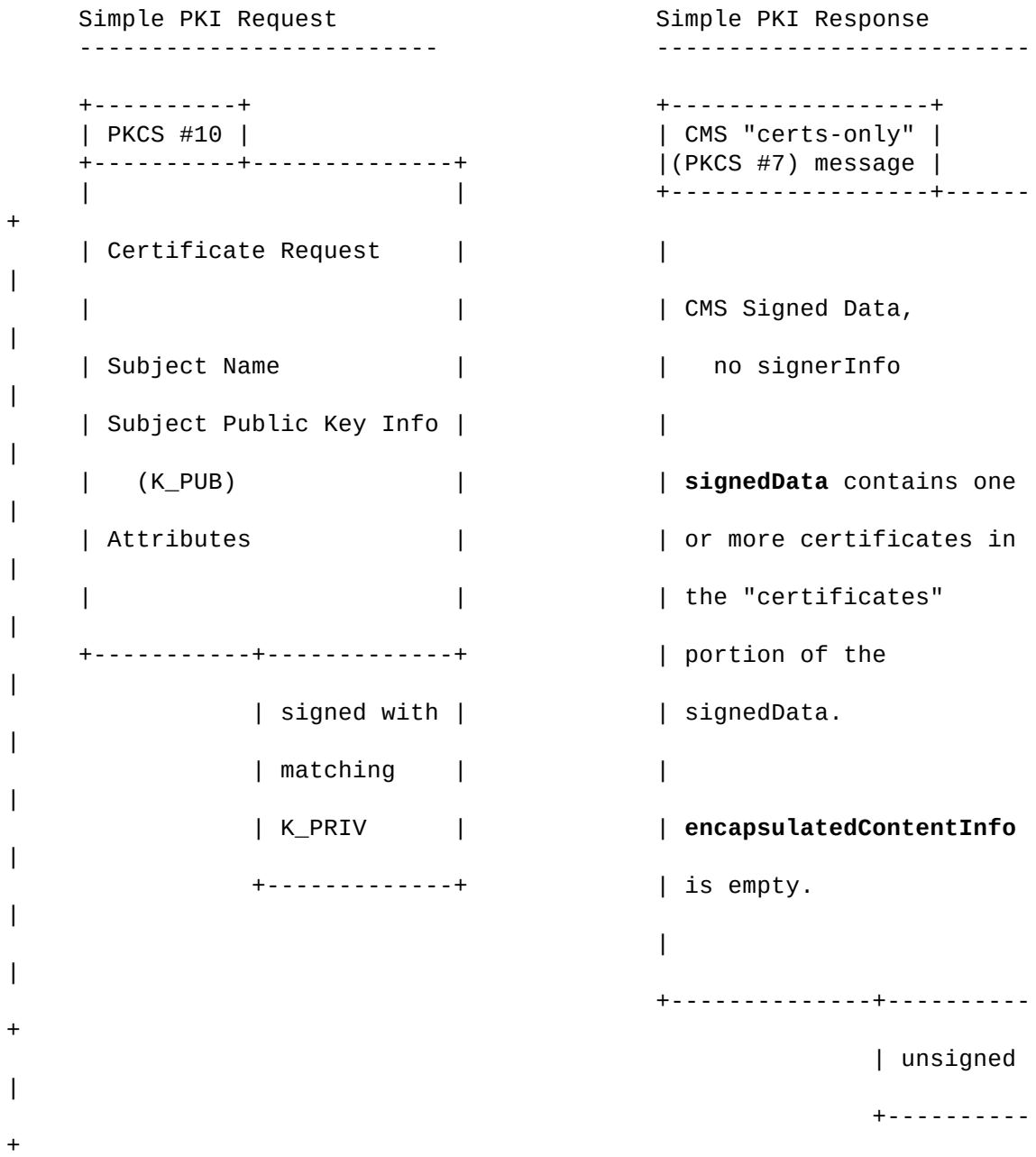
5 PKCS enrollment protocol

This section provides an overview of the PKI enrollment process using the PKCS standards. It provides a profile of the raw PKCS #10, Section 7, Reference 4, request message (Section 5.3.2) and the PKCS #7, Section 7, Reference 5, response message (Section 5.4.2) as well as their PKIX-CMP formats (Section 5.6.1 and Section 5.6.2, respectively) for PKI enrollment of subscribers and subject CAs.

5.1 Protocol Flow Charts

Figure 4 shows the Simple Enrollment Request and Response messages. The contents of these messages are detailed in Sections 5.2.1 and Section 5.2.2 below.

Figure 4. Simple PKI Request and Response Messages



5.2 PKI Messages

This section discusses the details of putting together the different enrollment request and response messages.

5.2.1 Simple Enrollment Request

The simplest form of an enrollment request is a plain PKCS #10 message. If this form of enrollment request is used for a private key that is capable of generating a signature, the PKCS #10 must be signed with that private key.

Servers must support the Simple Enrollment Request message. If the Simple Enrollment Request message is used, servers must return the Simple Enrollment Response message (see Section 5.2.2) if the enrollment request is granted. If the enrollment request fails, the Full Enrollment Response may be returned or no response may be returned.

5.2.2 Simple Enrollment Response

CAs should use the simple enrollment response message whenever possible. Clients must be able to process the simple enrollment response message. The simple enrollment response message consists of a **signedData** object with no **signerInfo** objects on it. The certificates requested are returned in the certificate bag of the **signedData** object.

Clients must not assume the certificates are in any order. Servers should include all intermediate certificates needed to form complete chains to one or more self-signed certificates, not just the newly issued certificate(s). Clients must not implicitly trust included self-signed certificate(s) merely due to its presence in the certificate bag. In the event clients receive a new self-signed certificate from the server, clients should provide a mechanism to enable the user to explicitly trust the certificate.

5.3 PKCS #10

Certification Requests are based on either PKCS #10 or CRMF, Section 7 Reference 6, (PKIX-CMP) messages. Section 5.6.1 specifies the PKCS #10 certification request message syntax. Section 3.4 describes the PKIX-CMP certification message flows.

5.3.1 General overview

The PKCS #10 standard describes a syntax for certification requests. A certification request consists of a distinguished name, a public key, and optionally a set of attributes, collectively signed by the entity requesting certification. Certification requests are sent to a CA, who transforms the request to an X.509 public-key certificate. A PKCS #7 message is a typical form for the CA to return the newly signed certificate. The preliminary intended application of the PKCS #10 standard is to support PKCS #7 cryptographic messages.

5.3.2 General syntax

When producing a PKCS #10 request body, clients must produce a PKCS #10 message body containing a subject name and public key. An entity would typically send a PKCS #10 certification request (which is then wrapped in a PKIX message) to the CA, for first time initialization, after generating a public-key/private-key pair, but may also do so after a change in the entity's DN. A reference number and authorization code must still be retrieved out-of-band by the client. As a result, the subject within the PKCS #10 message must be either the actual DN of the user associated with the given reference number or a DN of the form cn=<reference #>.

Some certification products are operated using a central repository of information to assign subject names upon receipt of a public key for certification. To accommodate this mode of operation, the subject name in a **CertificationRequest** may be null, but must be present. CAs that receive a **CertificationRequest** with a null subject name may reject such requests. If rejected and a response is returned, the CA must respond with the **failInfo** attribute of **badRequest**.

A PKCS #10 certification request consists of three parts:

- certification request information,
- a signature algorithm identifier, and
- a digital signature on the certification request information.

The certificate request structure (**CertificationRequest**) is defined in Section 5.3.2.1. The certificate request information structure (**CertificationRequestInfo**) is defined in Section 5.3.2.2.

5.3.2.1 CertificationRequest

A certification request shall have ASN.1 type **CertificationRequest**:

```
CertificationRequest ::= SEQUENCE {
        version                 0
        -- PKCS #10 version is currently 0
        subject                         User name,
        subjectPublicKeyInfo    SubjectPublicKeyInfo,
        -- Key information for the public verification key
        attributes              [0] IMPLICIT Attributes
        -- Must include keyUsage and extKeyUsage extension
        signatureAlgorithm      SignatureAlgorithmIdentifier
        -- e.g., md5WithRSAEncryption
        signature               Signature }
```

The **CertificationRequest** structure described above is constructed from the base **CertificationRequest** and **CertificationRequestInfo** structures, described below.

```
AlgorithmIdentifier  ::=  SEQUENCE  {
        Algorithm               OBJECT IDENTIFIER,
        Parameters              ANY DEFINED BY algorithm OPTIONAL  }

SubjectPublicKeyInfo  ::=  SEQUENCE  {
        Algorithm               AlgorithmIdentifier,
        SubjectPublicKey        BIT STRING  }

CertificationRequest ::= SEQUENCE {
        certificationRequestInfo        CertificationRequestInfo,
        signatureAlgorithm              SignatureAlgorithmIdentifier,
        signature                       Signature }

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier

Signature ::= BIT STRING
```

The fields of type **CertificationRequest** have the following meanings:

- **certificationRequestInfo** is the "certification request information." It is the value being signed;
- **signatureAlgorithm** identifies the signature algorithm (and any associated parameters) under which the certification-request information is signed (e.g., PKCS #1's **md2WithRSAEncryption** and **md5WithRSAEncryption**); and
- **signature** is the result of signing the certification request information with the certification request subject's private key.

5.3.2.2 CertificationRequestInfo

Certification request information shall have ASN.1 type **CertificationRequestInfo**:

```
CertificationRequestInfo ::= SEQUENCE {
        version                 0
        subject                         User name,
        subjectPublicKeyInfo    SubjectPublicKeyInfo,
```

```
        attributes              [0] IMPLICIT Attributes }

Version ::= INTEGER

AlgorithmIdentifier  ::=  SEQUENCE  {
        Algorithm               OBJECT IDENTIFIER,
        Parameters              ANY DEFINED BY algorithm OPTIONAL  }

SubjectPublicKeyInfo  ::=  SEQUENCE  {
        Algorithm               AlgorithmIdentifier,
        SubjectPublicKey        BIT STRING  }

Attributes ::= SET OF Attribute

Attribute ::= SEQUENCE {
        type            AttributeType,
        values          SET OF AttributeValue
        -- Must include keyUsage and extKeyUsage extension }

AttributeType ::=   OBJECT IDENTIFIER

AttributeValue ::=   ANY
```

The fields of type **CertificationRequestInfo** have the following meanings:

- **version** is the version number, for compatibility with future revisions of the PKCS #10 standard. It shall be 0 for this version of the PKCS #10 standard;
- **subject** is the distinguished name of the certificate subject (the entity whose public key is to be certified);
- **subjectPublicKeyInfo** contains information about the public key being certified. The information identifies the entity's public-key algorithm (and any associated parameters). The information also includes a bit-string representation of the entity's public key; and
- **attributes** is a set of attributes providing additional information about the subject of the certificate. The Key Usage extension is required as an attribute.

5.4 PKCS #7

The PKCS #7 standard describes a general syntax for data that may have cryptography applied to it, such as digital signatures and digital envelopes. A degenerate case of the syntax provides a means for disseminating certificates and CRLs. This is the case for the PKCS #7 response to the PKCS #10 request.

5.4.1 General overview

The PKCS #7 syntax admits recursion, so that, for example, one envelope can be nested inside another, or one party can sign some previously enveloped digital data. It also allows arbitrary attributes, such as signing time, to be authenticated along with the content of a message, and provides for other attributes such as countersignatures to be associated with a signature. A degenerate case of the syntax provides a means for disseminating certificates and CRLs.

The PKCS #7 syntax is general enough to support many different content types. The PKCS #7 standard defines six types:

- data,
- signed data,
- enveloped data,

- signed-and-enveloped data,
- digested data, and
- encrypted data.

For the PKCS #7 response to the PKCS #10 request, the signed-data and data content types are used.

5.4.2 General syntax

5.4.2.1 Overall ContentInfo

The general syntax for content exchanged between entities according to the PKCS #7 standard associates a content type with content. The syntax has ASN.1 type **ContentInfo**:

```
ContentInfo ::= SEQUENCE {
        contentType         signedData (1.2.840.113549.1.7.2)
        version             1
        -- This is 1 for this version of the PKCS #7 standard
        digestAlgorithms    DigestAlgorithmIdentifiers
        -- This is NULL
        contentType         data (1.2.840.113549.1.7.1)
        certificate
                version                 [0] Version,
                serialNumber            CertificateSerialNumber,
                signature               AlgorithmIdentifier,
                issuer                  Name,
                validity                Validity,
                subject                 Name,
                subjectPublicKeyInfo    SubjectPublicKeyInfo,
                issuerUniqueIdentifier  [1] IMPLICIT UniqueIdentifier OPTIONAL,
                -- This is NULL
                subjectUniqueIdentifier         [2] IMPLICIT UniqueIdentifier
                OPTIONAL
                -- This is NULL
                extensions              [3] Extensions OPTIONAL
                -- If present, version must be v3 }
        crls                [1] IMPLICIT CertificateRevocationLists OPTIONAL
        -- This is NULL
        signerInfos         SignerInfos
        -- This is NULL  }

AlgorithmIdentifier  ::=  SEQUENCE  {
        Algorithm               OBJECT IDENTIFIER,
        Parameters              ANY DEFINED BY algorithm OPTIONAL  }

SubjectPublicKeyInfo  ::=  SEQUENCE  {
        Algorithm               AlgorithmIdentifier,
        SubjectPublicKey        BIT STRING  }
```

The fields of type **ContentInfo** have the following meanings:

- **contentType** indicates the type of content. It is an OID, which means it is a unique string of integers assigned by the authority that defines the content type. For the PKCS #7 response to the PKCS #10 request, the content type is **SignedData**. Within the nested ContentInfo structure, this is **Data**.

- **content** is the content. The field is optional, and if the field is not present, its intended value must be supplied by other means. Its type is defined along with the object identifier for **contentType**.

5.4.2.2 Signed-data content type

The signed-data content type consists of content of any type and encrypted message digests of the content for zero or more signers. The encrypted digest for a signer is a "digital signature" on the content for that signer. Any type of content can be signed by any number of signers in parallel. It is expected that the typical application of the signed-data content type will be to represent one signer's digital signature on content of the data content type.

Note: The syntax has a degenerate case in which there are no signers on the content. The degenerate case provides a means for disseminating certificates and CRLs. This is the case for a response to the PKCS #10 request.

Note: In response to a PKCS #10 request, a typical application will be to disseminate certificates and CRLs.

The signed-data content type shall have ASN.1 type **SignedData**:

```
SignedData ::= SEQUENCE {
        version             1
        digestAlgorithms    DigestAlgorithmIdentifiers,
        contentInfo         ContentInfo,
        certificates        [0] IMPLICIT ExtendedCertificatesAndCertificates
OPTION,
        crls                [1] IMPLICIT CertificateRevocationLists OPTIONAL,
        signerInfos         SignerInfos }
```

The fields of type **SignedData** have the following meanings.

- **version** is the syntax version number. It is 1 for this version of the PKCS #7 standard.
- **digestAlgorithms** is a collection of message-digest algorithm identifiers. There may be any number of elements in the collection, including zero (**NULL**), which would be the case for the PKCS #7 response to the PKCS #10 request.

  Each element identifies the message-digest algorithm (and any associated parameters) under which the content is digested for a signer. The collection is intended to list the message-digest algorithms employed by all of the signers, in any order, to facilitate one-pass signature verification.
- **contentInfo** is the content that is signed. It can have any of the defined content types. For the PKCS #7 response to the PKCS #10 request, the content type is **data**.
- **certificates** is a set of X.509 certificates. It is intended that the set be sufficient to contain chains from a recognized "root" or "top-level CA" to all of the signers in the **signerInfos** field. There may be more certificates than necessary, and there may be certificates sufficient to contain chains from two or more independent top-level CAs. There may also be fewer certificates than necessary, if it is expected that those verifying the signatures have an alternate means of obtaining necessary certificates (e.g., from a previous set of certificates).
- **crls** is a set of CRLs. For the PKCS #7 response to the PKCS #10 request, the **crls** field is **NULL**.

It is intended that the set contain information sufficient to determine whether or not the certificates in the certificates field are "hot listed," but such correspondence is not necessary.

- **signerInfos**. There may be any number of elements in the collection, including zero, which would be the case for the PKCS #7 response to the PKCS #10 request.

  This is a collection of per-signer information.

```
ExtendedCertificatesAndCertificates ::= SET OF ExtendedCertificateOrCertificate

ExtendedCertificateOrCertificate ::= CHOICE {
        certificate             Certificate,
        extendedCertificate     [0] IMPLICIT ExtendedCertificate }

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

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

AlgorithmIdentifier ::= SEQUENCE {
        Algorithm               OBJECT IDENTIFIER,
        Parameters              ANY DEFINED BY algorithm OPTIONAL }

SubjectPublicKeyInfo ::= SEQUENCE {
        Algorithm               AlgorithmIdentifier,
        SubjectPublicKey        BIT STRING }
```

5.4.2.3 Nested ContentInfo

In the degenerate case where there are no signers on the content, the **ContentInfo** value being "signed" is irrelevant. It is recommended in that case that the content type of the **ContentInfo** value being "signed" be **Data**, and the content field of the **ContentInfo** value be omitted (**NULL**).

```
ContentInfo ::= SEQUENCE {
        contentType     ContentType,
        content                 [0] EXPLICIT ANY DEFINED BY contentType
OPTIONAL }
```

The fields of type **ContentInfo** have the following meanings:

- **contentType** indicates the type of content. It is an OID, which means it is a unique string of integers assigned by the authority that defines the content type. For the PKCS #7 response to the PKCS #10 request, the content type is **Data**; and

- **content** is the content. The field is optional, and if the field is not present, its intended value must be supplied by other means. Its type is defined along with the object identifier for **contentType**. For the PKCS #7 response to the PKCS #10 request, the content is **NULL**.

5.5 Message Flow Behavior

The signature on the certification request prevents an entity from requesting a certificate with another party's public key. Such an attack would give the entity the minor ability to pretend to be the originator of any message signed by the other party. This attack is significant only if the entity does not know the message being signed, and the signed part of the message does not identify the signer. The entity would still not be able to decrypt messages intended for the other party, of course.

A CA fulfills the request by verifying the entity's digital signature, and, if it is valid, constructing a X.509 certificate from the distinguished name and public key, as well as an issuer name, serial number, validity period, and signature algorithm of the CA's choice.

CAs are:

- required to be able to process all extensions defined in Section 7, Reference 7.
- not required to be able to process other X.509 v3 extensions transmitted using this protocol, nor are they required to be able to process other, private extensions;
- not required to put all client-requested extensions into a certificate; and
- permitted to modify client-requested extensions but not to alter an extension so as to invalidate the original intent of a client-requested extension (e.g., changing **keyUsage** from key exchange to signing). If a certification request is denied due to the inability to handle a requested extension and a response is returned, the CA must respond with the **failInfo** attribute of **unsupportedExt**.

5.6 Message Flows

CAs must be able to understand and process PKCS #10 request bodies. Clients must produce a PKCS #10 request body when using the Simple Enrollment Request message.

The message flow is the same as all requests which result in the generation of a certificate. Hence, a confirmation message from the client is required.

The PKCS #10 request exchange consists of three messages:

- PKCS #10 request (**pkcs10Req**),
- PKCS #7 response (**pkcs7Rep**), and
- Confirmation (**pkiConfirm**).

5.6.1 PKCS #10 request

pkcs10Req:: client ----------->> CA
{

pvno	1
sender	User name
recipient	CA name
messageTime	Current time
protectionAlg	**MSG_MAC_ALG**
senderKID	Reference number
transactionID	Implementation-specific (meaningful to client)
senderNonce	128-bit pseudo-random number

freeText	Text
body	**CertificationRequest**
protection	bits calculated using **MSG_MAC_ALG**

} MAC (key based on authorization code is used to create MAC for structure)

The message is verified by the CA. If valid, the CA generates a new certificate for the client and sends a modifyRequest message to the directory to issue a new certificate. The CA responds with a **pkcs7Rep** message to the client.

The PKCS # 10 **CertificationRequest** structure is defined in Section 5.3.2.1.

5.6.2 PKCS #7 response

pkcs7Rep:: CA ----------->> client
{

pvno	1
sender	CA name
recipient	User name
messageTime	Time at which CA produced message
protectionAlg	**MSG_MAC_ALG**
recipKID	Reference number
transactionID	Value from corresponding **pkcs10Req** message
senderNonce	Value from corresponding **pkcs10Req** message
recipNonce	128-bit pseudo-random number
freeText	Text
body	**CertificationResponse**
certificate	Present
privateKey	Present
protection	bits calculated using **MSG_MAC_ALG**
extraCerts	CA's protocol verification certificate

} MAC (key based on authorization code is used to create MAC for structure)

The message is verified by the client. It then sends a **pKIConfirm** message to CA. The PKCS # 7 **CertificationResponse** structure is defined in Section 5.4.2.1.

5.6.3 Confirmation

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

pkiConfirm:: client ----------->> CA
{

pvno	1
sender	User name
recipient	CA name
transactionID	Value from corresponding **pkcs10Req** message
senderNonce	Value from corresponding **pkcs10Req** message
recipNonce	Value from corresponding **pkcs7Rep** message
protectionAlg	**MSG_MAC_ALG**
senderKID	For verification of message protection
freeText	Text
protection	bits calculated using **MSG_MAC_ALG**

} MAC (key based on authorization code is used to create MAC for structure)

6 Glossary

ARL	Authority Revocation List
ASN.1	Abstract Syntax Notation 1
CA	Certification Authority
CRL	Certificate Revocation List
DER	Distinguished Encoding Rules
DN	Distinguished Name
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
EE	End Entity
GUI	Graphical User Interface
IETF	Internet Engineering Task Force
IPSec	Internet Protocol Security
ITU-T	International Telecommunications Union Telecommunications Sector
LDAP	Lightweight Directory Access Protocol
MD2, MD5	Message Digest 2, 5
OID	Object Identifier
PEM	Privacy Enhanced Mail
PKI	Public Key Infrastructure
PKIX	Public Key Infrastructure X.509 based
PSE	Personal Security Environment
RA	Registration Authority
RSA	Rivest, Shamir, Adelman (RSA algorithm)
SHA-1	Secure Hash Algorithm 1
TSA	Time Stamp Authority
TTP	Trusted Third Party

7 References

[Reference 1] Internet Draft. Internet X.509 Public Key Infrastructure. Time Stamp Protocol <draft-ietf-pkix-time-stamp-04.txt>. C. Adams, P. Cain, D. Pinkas, R. Zuccherato. October 1999.

[Reference 2] ISO 9594-8:1995. Information Technology - Open Systems Interconnection - The Directory: Authentication Framework. 1995.

[Reference 3] RFC 2510. Internet X.509 Public Key Infrastructure Certificate Management Protocols (CMP). March 1999.

[Reference 4] PKCS #10: Certification Request Syntax Standard. RSA Laboratories. Version 1.0. November 1, 1993.

[Reference 5] PKCS #7: Cryptographic Message Syntax Standard. RSA Laboratories. Version 1.5. Revised November 1, 1993.

[Reference 6] RFC 2511. Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF). March 1999.

[Reference 7] RFC 2459. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. R. Housley, W. Ford, W. Polk, and D. Solo. January 1999.