Network Working Group
Request for Comments: 2797
Category: Standards Track

M. Myers
VeriSign
X. Liu
Cisco
J. Schaad
Microsoft
J. Weinstein
April 2000

Certificate Management Messages over CMS

Status of this Memo

Copyright Notice

Abstract

This document defines a Certificate Management protocol using CMS (CMC). This protocol addresses two immediate needs within the Internet PKI community:

1. The need for an interface to public key certification products and services based on [CMS] and [PKCS10], and
2. The need in [SMIMEV3] for a certificate enrollment protocol for DSA-signed certificates with Diffie-Hellman public keys.

A small number of additional services are defined to supplement the core certificate request service.

Throughout this specification the term CMS is used to refer to both [CMS] and [PKCS7]. For both signedData and envelopedData, CMS is a superset of the PKCS7. In general, the use of PKCS7 in this document is aligned to the Cryptographic Message Syntax [CMS] that provides a superset of the PKCS7 syntax. The term CMC refers to this specification.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

1. Protocol Requirements

- The protocol is to be based as much as possible on the existing CMS, PKCS#10 and CRMF specifications.
- The protocol must support the current industry practice of a PKCS#10 request followed by a PKCS#7 response as a subset of the protocol.
- The protocol needs to easily support the multi-key enrollment protocols required by S/MIME and other groups.
- The protocol must supply a way of doing all operations in a single-round trip. When this is not possible the number of round trips is to be minimized.
- The protocol will be designed such that all key generation can occur on the client.
- The mandatory algorithms must superset the required algorithms for S/MIME.
- The protocol will contain POP methods. Optional provisions for multiple-round trip POP will be made if necessary.
- The protocol will support deferred and pending responses to certificate request for cases where external procedures are required to issue a certificate.
- The protocol needs to support arbitrary chains of local registration authorities as intermediaries between certificate requesters and issuers.

2. Protocol Overview

An enrollment transaction in this specification is generally composed of a single round trip of messages. In the simplest case an enrollment request is sent from the client to the server and an enrollment response is then returned from the server to the client. In some more complicated cases, such as delayed certificate issuance and polling for responses, more than one round trip is required.

This specification supports two different request messages and two different response messages.

Public key certification requests can be based on either the PKCS10 or CRMF object. The two different request messages are (a) the bare PKCS10 (in the event that no other services are needed), and (b) the PKCS10 or CRMF message wrapped in a CMS encapsulation as part of a PKIData object.

Public key certification responses are based on the CMS signedData object. The response may be either (a) a degenerate CMS signedData object (in the event no other services are needed), or (b) a ResponseBody object wrapped in a CMS signedData object.

No special services are provided for doing either renewal (new certificates with the same key) or re-keying (new certificates on new keys) of clients. Instead a renewal/re-key message looks the same as any enrollment message, with the identity proof being supplied by existing certificates from the CA.

A provision exists for Local Registration Authorities (LRAs) to participate in the protocol by taking client enrollment messages, wrapping them in a second layer of enrollment message with additional requirements or statements from the LRA and then passing this new expanded request on to the Certification Authority.

This specification makes no assumptions about the underlying transport mechanism. The use of CMS is not meant to imply an email-based transport.

Optional services available through this specification are transaction management, replay detection (through nonces), deferred certificate issuance, certificate revocation requests and certificate/CRL retrieval.

2.1 Terminology

There are several different terms, abbreviations and acronyms used in this document that we define here for convenience and consistency of usage:

"End-Entity" (EE) refers to the entity that owns a key pair and for whom a certificate is issued.
"LRA" or "RA" refers to a (Local) Registration Authority. A registration authority acts as an intermediary between an End-Entity and a Certification Authority. Multiple RAs can exist between the End-Entity and the Certification Authority.
"CA" refers to a Certification Authority. A Certification Authority is the entity that performs the actual issuance of a certificate.
"Client" refers to an entity that creates a PKI request. In this document both RAs and End-Entities can be clients.
"Server" refers to the entities that process PKI requests and create PKI responses. CAs and RAs can be servers in this document.
"PKCS#10" refers the Public Key Cryptography Standard #10. This is one of a set of standards defined by RSA Laboratories in the 1980s. PKCS#10 defines a Certificate Request Message syntax.
"CRMF" refers to the Certificate Request Message Format RFC [CRMF]. We are using certificate request message format defined in this document as part of our management protocol.
"CMS" refers to the Cryptographic Message Syntax RFC [CMS]. This document provides for basic cryptographic services including encryption and signing with and without key management.

"POP" is an acronym for "Proof of Possession". POP refers to a value that can be used to prove that the private key corresponding to a public key is in the possession and can be used by an end-entity. "Transport wrapper" refers to the outermost CMS wrapping layer.

2.2 Protocol Flow Charts

Figure 1 shows the Simple Enrollment Request and Response messages. The contents of these messages are detailed in Sections 4.1 and 4.3 below.

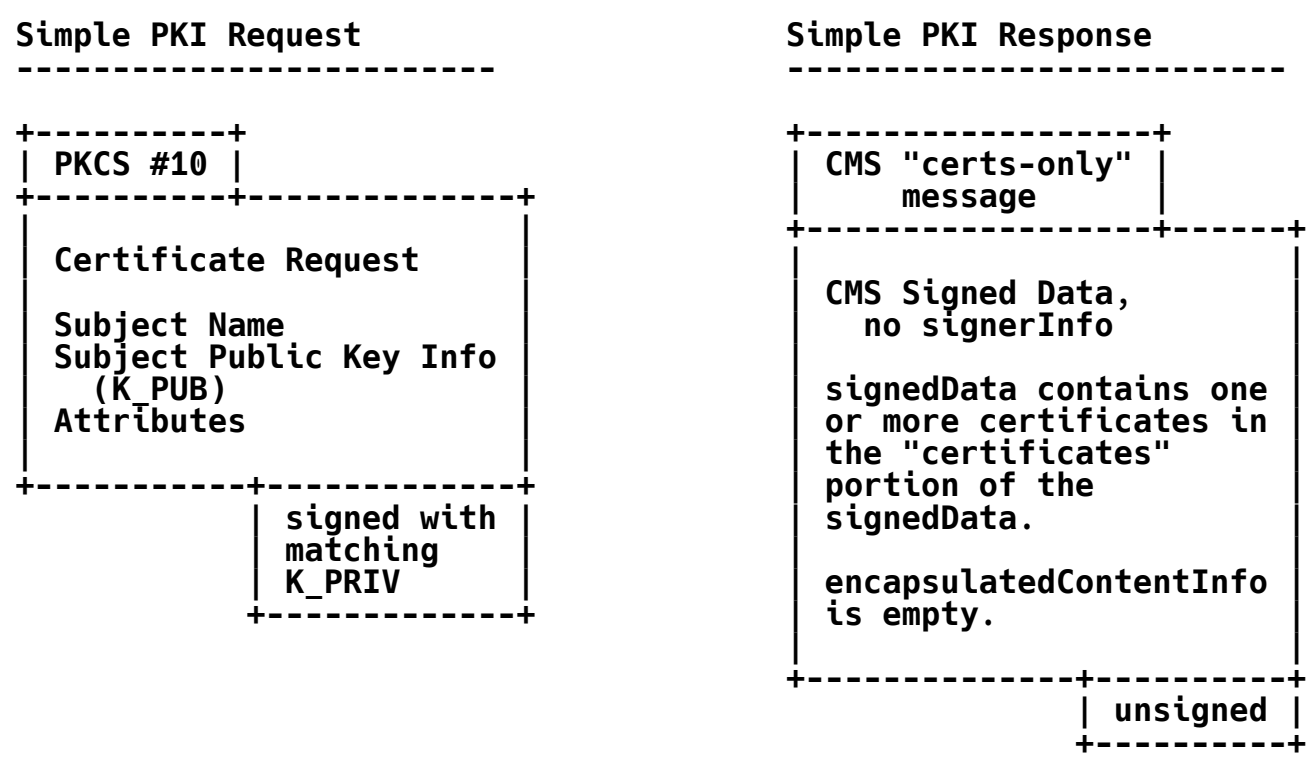

Figure 1: Simple PKI Request and Response Messages

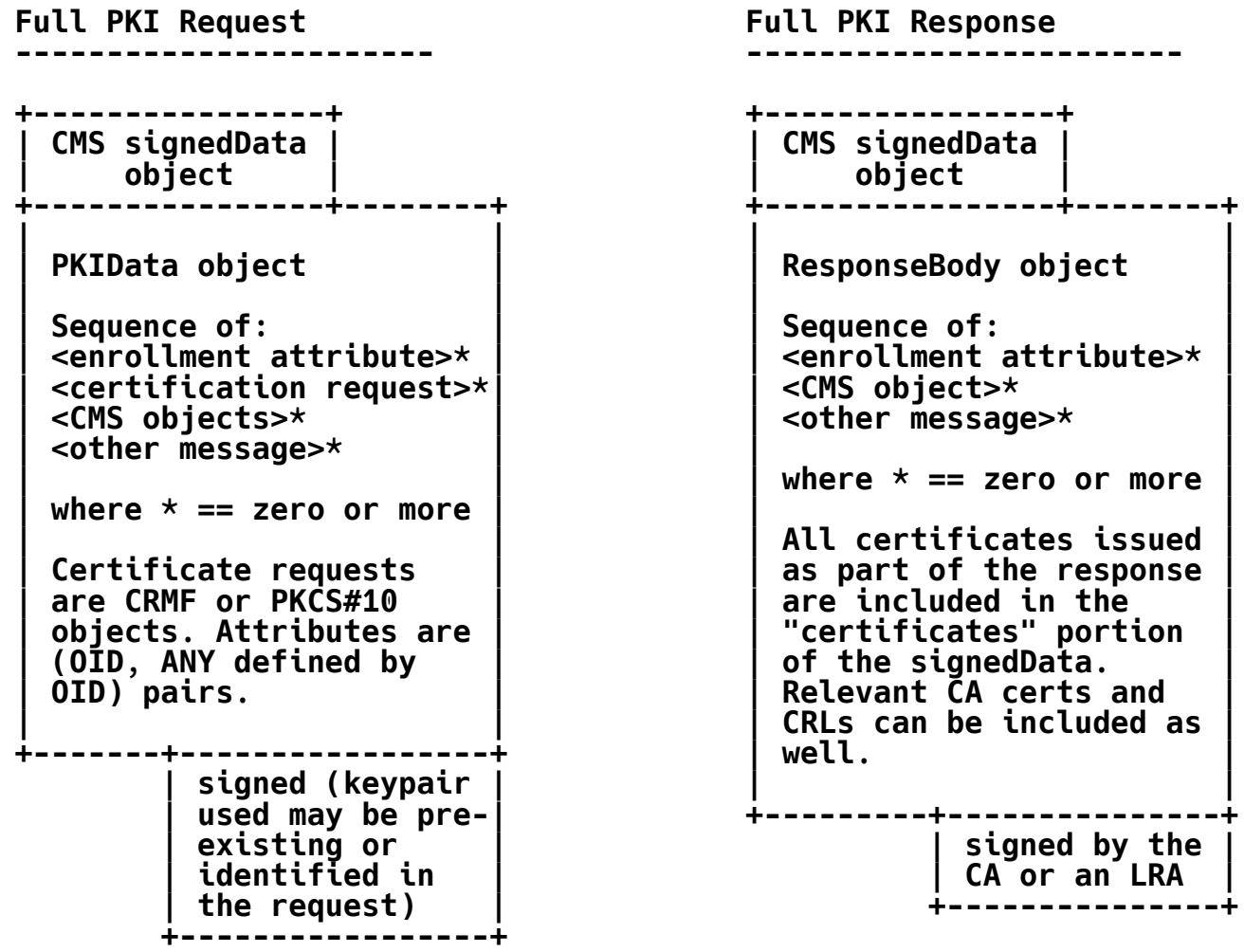

Figure 2: Full PKI Request and Response Messages

Figure 2 shows the Full Enrollment Request and Response messages. The contents of these messages are detailed in Sections 4.2 and 4.4 below.

3. Protocol Elements

This section covers each of the different elements that may be used to construct enrollment request and enrollment response messages. Section 4 will cover how to build the enrollment request and response messages.

3.1 PKIData Object

The new content object PKIData has been defined for this protocol. This new object is used as the body of the full PKI request message. The new body is identified by:

```
  id-cct-PKIData  OBJECT IDENTIFIER ::= { id-cct 2 }
```

The ASN.1 structure corresponding to this new content type is:

```
PKIData ::= SEQUENCE {
      controlSequence    SEQUENCE SIZE(0..MAX) OF TaggedAttribute,
      reqSequence        SEQUENCE SIZE(0..MAX) OF TaggedRequest,
      cmsSequence        SEQUENCE SIZE(0..MAX) OF TaggedContentInfo,
      otherMsgSequence   SEQUENCE SIZE(0..MAX) OF OtherMsg
}

-- controlSequence consists of a sequence of control attributes.  The
control attributes defined in this document are found in section 5.
As control sequences are defined by OIDs, other parties can define
additional control attributes. Unrecognized OIDs MUST result in no
part of the request being successfully processed.

-- reqSequence consists of a sequence of certificate requests.  The
certificate requests can be either a CertificateRequest (PKCS10
request) or a CertReqMsg.  Details on each of these request types are
found in sections 3.3.1 and 3.3.2 respectively.

-- cmsSequence consists of a sequence of [CMS] message objects.  This
protocol only uses EnvelopedData, SignedData and EncryptedData.  See
section 3.6 for more details.

-- otherMsgSequence allows for other arbitrary data items to be
placed into the enrollment protocol.  The {OID, any} pair of values
allows for arbitrary definition of material.  Data objects are placed
here while control objects are placed in the controlSequence field.
See section 3.7 for more details.
```

3.2 ResponseBody Object

The new content object ResponseBody has been defined for this protocol. This new object is used as the body of the full PKI response message. The new body is identified by:

```
    id-cct-PKIResponse OBJECT IDENTIFIER ::= { id-cct 3 }
```

The ASN.1 structure corresponding to this body content type is:

```
ResponseBody ::= SEQUENCE {
    controlSequence   SEQUENCE SIZE(0..MAX) OF TaggedAttribute,
    cmsSequence       SEQUENCE SIZE(0..MAX) OF TaggedContentInfo,
    otherMsgSequence  SEQUENCE SIZE(0..MAX) OF OtherMsg
}

-- controlSequence consists of a sequence of control attributes.  The
control attributes defined in this document are found in section 3.5.
Other parties can define additional control attributes.

-- cmsSequence consists of a sequence of [CMS] message objects.  This
protocol only uses EnvelopedData, SignedData and EncryptedData.  See
section 3.6 for more details.

-- otherMsgSequence allows for other arbitrary items to be placed
into the enrollment protocol.  The {OID, any} pair of values allows
for arbitrary definition of material.  Data objects are placed here
while control objects are placed in the controlSequence field. See
section 3.7 for more details.
```

3.3 Certification Requests (PKCS10/CRMF)

Certification Requests are based on either PKCS10 or CRMF messages. Section 3.3.1 specifies mandatory and optional requirements for clients and servers dealing with PKCS10 request messages. Section 3.3.2 specifies mandatory and optional requirements for clients and servers dealing with CRMF request messages.

3.3.1 PKCS10 Request Body

Servers MUST be able to understand and process PKCS10 request bodies. Clients MUST produce a PKCS10 request body when using the Simple Enrollment Request message. Clients MAY produce a PKCS10 request body when using the Full Enrollment Request message.

When producing a PKCS10 request body, clients MUST produce a PKCS10 message body containing a subject name and public key. 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.

The client MAY incorporate one or more standard X.509 v3 extensions in any PKCS10 request as an ExtensionReq attribute. An ExtensionReq attribute is defined as

```
ExtensionReq ::= SEQUENCE OF Extension
```

where Extension is imported from [PKIXCERT] and ExtensionReq is identified by {pkcs-9 14}.

Servers MUST be able to process all extensions defined in [PKIXCERT]. Servers are not required to be able to process other V3 X.509 extensions transmitted using this protocol, nor are they required to be able to process other, private extensions. Servers are not required to put all client-requested extensions into a certificate. Servers are permitted to modify client-requested extensions. Servers MUST NOT alter an extension so as to invalidate the original intent of a client-requested extension. (For example changing key usage 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 server MUST respond with the failInfo attribute of unsupportedExt.

3.3.2 CRMF Request Body

Servers MUST be able to understand and process CRMF request body. Clients MAY produce a CRMF message body when using the Full Enrollment Request message.

This memo imposes the following additional changes on the construction and processing of CRMF messages:

- When CRMF message bodies are used in the Full Enrollment Request message, each CRMF message MUST include both the subject and publicKey fields in the CertTemplate. As in the case of PKCS10 requests, the subject may be encoded as NULL, but MUST be present.
- In general, when both CRMF and CMC controls exist with equivalent functionality, the CMC control SHOULD be used. The CMC control MUST override any CRMF control.
- The regInfo field MUST NOT be used on a CRMF message. Equivalent functionality is provided in the regInfo control attribute (section 5.12).
- The indirect method of proving POP is not supported in this protocol. One of the other methods (including the direct method described in this document) MUST be used instead if POP is desired. The value of encrCert in SubsequentMessage MUST NOT be used.

- Since the subject and publicKeyValues are always present, the POPOSigningKeyInput MUST NOT be used when computing the value for POPSigningKey.

A server is not required to use all of the values suggested by the client in the certificate template. Servers MUST be able to process all extensions defined in [PXIXCERT]. Servers are not required to be able to process other V3 X.509 extension transmitted using this protocol, nor are they required to be able to process other, private extensions. Servers are permitted to modify client-requested extensions. Servers MUST NOT alter an extension so as to invalidate the original intent of a client-requested extension. (For example change key usage from key exchange to signing.) If a certificate request is denied due to the inability to handle a requested extension, the server MUST respond with a failInfo attribute of unsupportedExt.

3.3.3 Production of Diffie-Hellman Public Key Certification Requests

Part of a certification request is a signature over the request; Diffie-Hellman is a key agreement algorithm and cannot be used to directly produce the required signature object. [DH-POP] provides two ways to produce the necessary signature value. This document also defines a signature algorithm that does not provide a POP value, but can be used to produce the necessary signature value.

3.3.3.1 No-Signature Signature Mechanism

Key management (encryption/decryption) private keys cannot always be used to produce some type of signature value as they can be in a decrypt only device. Certification requests require that the signature field be populated. This section provides a signature algorithm specifically for that purposes. The following object identifier and signature value are used to identify this signature type:

```
id-alg-noSignature OBJECT IDENTIFIER ::= {id-pkix id-alg(6) 2}

NoSignatureValue ::= OCTET STRING
```

The parameters for id-alg-noSignature MUST be present and MUST be encoded as NULL. NoSignatureValue contains the hash of the certification request. It is important to realize that there is no security associated with this signature type. If this signature type is on a certification request and the Certification Authority policy requires proof-of-possession of the private key, the POP mechanism defined in section 5.7 MUST be used.

3.3.3.2 Diffie-Hellman POP Signature

CMC compliant implementations MUST support section 5 of [DH-POP].

3.3.3.3 Diffie-Hellman MAC signature

CMC compliant implementations MAY support section 4 of [DH-POP].

3.4 Body Part Identifiers

Each element of a PKIData or PKIResponse message has an associated body part identifier. The Body Part Identifier is a 4-octet integer encoded in the certReqIds field for CertReqMsg objects (in a TaggedRequest) or in the bodyPartId field of the other objects. The Body Part Identifier MUST be unique within a single PKIData or PKIResponse object. Body Part Identifiers can be duplicated in different layers (for example a CMC message embedded within another). The Body Part Id of zero is reserved to designate the current PKIData object. This value is used in control attributes such as the Add Extensions Control in the pkiDataReference field to refer to a request in the current PKIData object.

Some control attribute, such as the CMC Status Info attribute, will also use Body Part Identifiers to refer to elements in the previous message. This allows an error to be explicit about the attribute or request to which the error applies.

3.5 Control Attributes

The overall control flow of how a message is processed in this document is based on the control attributes. Each control attribute consists of an object identifier and a value based on the object identifier.

Servers MUST fail the processing of an entire PKIData message if any included control attribute is not recognized. The response MUST be the error badRequest and bodyList MUST contain the bodyPartID of the invalid or unrecognized control attribute.

The syntax of a control attribute is

```
TaggedAttribute ::= SEQUENCE {
    bodyPartID          BodyPartId,
    attrType            OBJECT IDENTIFIER,
    attrValues          SET OF AttributeValue
}
```

```
      -- bodyPartId is a unique integer that is used to reference this
      control attribute. The id of 0 is reserved for use as the
      reference to the current PKIData object.

      -- attrType is the OID defining the associated data in attrValues

      -- attrValues contains the set of data values used in processing
      the control attribute.
```

The set of control attributes that are defined by this memo are found in section 5.

3.6 Content Info objects

The cmsSequence field of the PKIRequest and PKIResponse messages contains zero or more tagged content info objects. The syntax for this structure is

```
  TaggedContentInfo ::= SEQUENCE {
      bodyPartID              BodyPartId,
      contentInfo             ContentInfo
  }

      -- bodyPartId is a unique integer that is used to reference this
      content info object. The id of 0 is reserved for use as the
      reference to the current PKIData object.

      -- contentInfo contains a ContentInfo object (defined in [CMS]).
      The three contents used in this location are SignedData,
      EnvelopedData and Data.
```

EnvelopedData provides for shrouding of data. Data allows for general transport of unstructured data.

The SignedData object from [CMS] is also used in this specification to provide for authentication as well as serving as the general transport wrapper of requests and responses.

3.6.1 Signed Data

The signedData object is used in two different locations when constructing enrollment messages. The signedData object is used as a wrapper for a PKIData as part of the enrollment request message. The signedData object is also used as the outer part of an enrollment response message.

For the enrollment response the signedData wrapper allows the server to sign the returning data, if any exists, and to carry the certificates and CRLs for the enrollment request. If no data is being returned beyond the certificates, no signerInfo objects are placed in the signedData object.

3.6.2 Enveloped Data

EnvelopedData is the primary method of providing confidentiality for sensitive information in this protocol. The protocol currently uses EnvelopedData to provide encryption of an entire request (see section 4.5). The envelopedData object would also be used to wrap private key material for key archival.

Servers MUST implement envelopedData according to [CMS]. There is an ambiguity (about encrypting content types other than id-data) in the PKCS7 specification that has lead to non-interoperability.

3.7 Other Message Bodies

The other message body portion of the message allows for arbitrary data objects to be carried as part of a message. This is intended to contain data that is not already wrapped in a CMS contentInfo object. The data is ignored unless a control attribute references the data by bodyPartId.

```
  OtherMsg ::= SEQUENCE {
      bodyPartID        BodyPartID,
      otherMsgType      OBJECT IDENTIFIER,
      otherMsgValue     ANY DEFINED BY otherMsgType }
```

-- bodyPartID contains the unique id of this object

-- otherMsgType contains the OID defining both the usage of this body part and the syntax of the value associated with this body part

-- otherMsgValue contains the data associated with the message body part.

4. PKI Messages

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

4.1 Simple Enrollment Request

The simplest form of an enrollment request is a plain PKCS10 message. If this form of enrollment request is used for a private key that is capable of generating a signature, the PKCS10 MUST be signed with that private key. If this form of the enrollment request is used for a D-H key, then the D-H POP mechanism described in [DH-POP] MUST be used.

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 4.3) if the enrollment request is granted. If the enrollment request fails, the Full Enrollment Response MAY be returned or no response MAY be returned.

Many advanced services specified in this memo are not supported by the Simple Enrollment Request message.

4.2 Full PKI Request

The Full Enrollment Request provides the most functionality and flexibility. Clients SHOULD use the Full Enrollment Request message when enrolling. Servers MUST support the Full Enrollment Request message. An enrollment response (full or simple as appropriate) MUST be returned to all Full Enrollment Requests.

The Full Enrollment Request message consists of a PKIData object wrapped in a signedData CMS object. The objects in the PKIData are ordered as follows:

1. All Control Attributes,
2. All certification requests,
3. All CMS objects,
4. All other messages.

Each element in a Full Enrollment Request is identified by a Body Part Identifier. If duplicate ids are found, the server MUST return the error badRequest with a bodyPartID of 0.

The signedData object wrapping the PKIData may be signed either by the private key material of the signature certification request, or by a previously certified signature key. If the private key of a signature certification request is being used, then:

a) the certification request containing the corresponding public key MUST include a Subject Key Identifier extension request,
b) the subjectKeyIdentifier form of signerInfo MUST be used, and

c) the value of the subjectKeyIdentifier form of signerInfo MUST be the Subject Key Identifier specified in the corresponding certification request.

(The subjectKeyIdentifier form of signerInfo is used here because no certificates have yet been issued for the signing key.) If the request key is used for signing, there MUST be only one signerInfo object in the signedData object.

When creating a message to renew a certificate, the following should be taken into consideration:

1. The identification and identityProof control statements are not required. The same information is provided by the use of an existing certificate from the CA when signing the enrollment message.
2. CAs and LRAs may impose additional restrictions on the signing certificate used. They may require that the most recently issued signing certificate for an entity be used.
3. A renewal message may occur either by creating a new set of keys, or by re-using an existing set of keys. Some CAs may prevent re-use of keys by policy. In this case the CA MUST return NOKEYREUSE as the failure code.

4.3 Simple Enrollment Response

Servers 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). The server MAY additionally return CRLs in the CRL bag. Servers MAY include the self-signed certificates. 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.

4.4 Full PKI Response

Servers MUST return full PKI response messages if a) a full PKI request message failed or b) additional services other than returning certificates are required. Servers MAY return full PKI responses with failure information for simple PKI requests. Following section 4.3 above, servers returning only certificates and a success status to the client SHOULD use the simple PKI response message.

Clients MUST be able to process a full PKI response message.

The full enrollment response message consists of a signedData object encapsulating a responseBody object. In a responseBody object all Control Attributes MUST precede all CMS objects. The certificates granted in an enrollment response are returned in the certificates field of the immediately encapsulating signedData object.

Clients MUST NOT assume the certificates are in any order. Servers SHOULD include all intermediate certificates needed to form complete chains one ore more self-signed certificates, not just the newly issued certificate(s). The server MAY additionally return CRLs in the CRL bag. Servers MAY include the self-signed certificates. 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.

4.5 Application of Encryption to a PKI Message

There are occasions where a PKI request or response message must be encrypted in order to prevent any information about the enrollment from being accessible to unauthorized entities. This section describes the means used to encrypt a PKI message. This section is not applicable to a simple enrollment message.

Confidentiality is provided by wrapping the PKI message (a signedData object) in a CMS EnvelopedData object. The nested content type in the EnvelopedData is id-signedData. Note that this is different from S/MIME where there is a MIME layer placed between the encrypted and signed data objects. It is recommended that if an enveloped data layer is applied to a PKI message, a second signing layer be placed outside of the enveloped data layer. The following figure shows how this nesting would be done:

Normal	Option 1	Option 2
SignedData	EnvelopedData	SignedData
PKIData	SignedData	EnvelopedData
	PKIData	SignedData
		PKIData

Options 1 and 2 provide the benefit of preventing leakage of sensitive data by encrypting the information. LRAs can remove the enveloped data wrapping, and replace or forward without further processing. Section 6 contains more information about LRA processing.

PKI Messages MAY be encrypted or transmitted in the clear. Servers MUST provided support for all three versions.

Alternatively, an authenticated, secure channel could exist between the parties requiring encryption. Clients and servers MAY use such channels instead of the technique described above to provide secure, private communication of PKI request and response messages.

5. Control Attributes

Control attributes are carried as part of both PKI requests and responses. Each control attribute is encoded as a unique Object Identifier followed by that data for the control attribute. The encoding of the data is based on the control attribute object identifier. Processing systems would first detect the OID and process the corresponding attribute value prior to processing the message body.

The following table lists the names, OID and syntactic structure for each of the control attributes documented in this memo.

Control Attribute	OID	Syntax
cMCStatusInfo	id-cmc 1	CMCStatusInfo
identification	id-cmc 2	UTF8String
identityProof	id-cmc 3	OCTET STRING
dataReturn	id-cmc 4	OCTET STRING
transactionId	id-cmc 5	INTEGER
senderNonce	id-cmc 6	OCTET STRING
recipientNonce	id-cmc 7	OCTET STRING
addExtensions	id-cmc 8	AddExtensions
encryptedPOP	id-cmc 9	EncryptedPOP
decryptedPOP	id-cmc 10	DecryptedPOP
lraPOPWitness	id-cmc 11	LraPOPWitness
getCert	id-cmc 15	GetCert
getCRL	id-cmc 16	GetCRL
revokeRequest	id-cmc 17	RevokeRequest
regInfo	id-cmc 18	OCTET STRING
responseInfo	id-cmc 19	OCTET STRING
QueryPending	id-cmc 21	OCTET STRING
idPOPLinkRandom	id-cmc 22	OCTET STRING
idPOPLinkWitness	id-cmc 23	OCTET STRING
idConfirmCertAcceptance	id-cmc 24	CMCCertId

5.1 CMC Status Info Control Attribute

The CMC status info control is used in full PKI Response messages to return information on a client request. Servers MAY emit multiple CMC status info controls referring to a single body part. Clients MUST be able to deal with multiple CMC status info controls in a response message. This statement uses the following ASN.1 definition:

```
CMCStatusInfo ::= SEQUENCE {
     cMCStatus             CMCStatus,
     bodyList              SEQUENCE SIZE (1..MAX) OF BodyPartID,
     statusString          UTF8String OPTIONAL,
     otherInfo             CHOICE {
       failInfo              CMCFailInfo,
       pendInfo              PendInfo } OPTIONAL
}

PendInfo ::= SEQUENCE {
     pendToken             OCTET STRING,
     pendTime              GeneralizedTime
}
```

```
      -- cMCStatus is described in section 5.1.1

      -- bodyList contains the list of body parts in the request message
      to which this status information applies.  If an error is being
      returned for a simple enrollment message, body list will contain a
      single integer of value '1'.

      -- statusString contains a string with additional description
      information.  This string is human readable.

      -- failInfo is described in section 5.1.2. It provides a detailed
      error on what the failure was.  This choice is present only if
      cMCStatus is failed.

      -- pendToken is the token to be used in the queryPending control
      attribute.

      -- pendTime contains the suggested time the server wants to be
      queried about the status of the request.
```

If the cMCStatus field is success, the CMC Status Info Control MAY be omitted unless it is only item in the response message. If no status exists for a certificate request or other item requiring processing, then the value of success is to be assumed.

5.1.1 CMCStatus values

CMCStatus is a field in the CMCStatusInfo structure. This field contains a code representing the success or failure of a specific operation. CMCStatus has the ASN.1 structure of:

```
   CMCStatus ::= INTEGER {
        success                 (0),
        -- request was granted
        -- reserved             (1),
        -- not used, defined where the original structure was defined
        failed                  (2),
        -- you don't get what you want, more information elsewhere in
   the message
        pending                 (3),
        -- the request body part has not yet been processed,
        -- requester is responsible to poll back on this
        -- pending may only be return for certificate request
   operations.
        noSupport               (4),
        -- the requested operation is not supported
        confirmRequired         (5)
```

```
          -- conformation using the idConfirmCertAcceptance control is
      required
          -- before use of certificate
      }
```

5.1.2 CMCFailInfo

CMCFailInfo conveys information relevant to the interpretation of a failure condition. The CMCFailInfo has the following ASN.1 structure:

```
      CMCFailInfo ::= INTEGER {
          badAlg            (0)
          -- Unrecognized or unsupported algorithm
          badMessageCheck   (1)
          -- integrity check failed
          badRequest        (2)
          -- transaction not permitted or supported
          badTime           (3)
          -- Message time field was not sufficiently close to the system
      time
          badCertId         (4)
          -- No certificate could be identified matching the provided
      criteria
          unsuportedExt     (5)
          -- A requested X.509 extension is not supported by the
      recipient CA.
          mustArchiveKeys   (6)
          -- Private key material must be supplied
          badIdentity       (7)
          -- Identification Attribute failed to verify
          popRequired       (8)
          -- Server requires a POP proof before issuing certificate
          popFailed         (9)
          -- POP processing failed
          noKeyReuse        (10)
          -- Server policy does not allow key re-use
          internalCAError   (11)
          tryLater          (12)
      }
```

Additional failure reasons MAY be defined for closed environments with a need.

5.2 Identification and IdentityProof Control Attributes

Some CAs and LRAs require that a proof of identity be included in a certification request. Many different ways of doing this exist with different degrees of security and reliability. Most people are familiar with the request of a bank to provide your mother's maiden name as a form of identity proof.

CMC provides one method of proving the client's identity based on a shared secret between the certificate requestor and the verifying authority. If clients support full request messages, clients MUST implement this method of identity proof. Servers MUST provide this method and MAY also have a bilateral method of similar strength available.

The CMC method starts with an out-of-band transfer of a token (the shared secret). The distribution of this token is beyond the scope of this document. The client then uses this token for an identity proof as follows:

1. The reqSequence field of the PKIData object (encoded exactly as it appears in the request message including the sequence type and length) is the value to be validated.
2. A SHA1 hash of the token is computed.
3. An HMAC-SHA1 value is then computed over the value produced in Step 1, as described in [HMAC], using the hash of the token from Step 2 as the shared secret value.
4. The 160-bit HMAC-SHA1 result from Step 3 is then encoded as the value of the identityProof attribute.

When the server verifies the identityProof attribute, it computes the HMAC-SHA1 value in the same way and compares it to the identityProof attribute contained in the enrollment request.

If a server fails the verification of an identityProof attribute and the server returns a response message, the failInfo attribute MUST be present in the response and MUST have a value of badIdentity.

Optionally, servers MAY require the inclusion of the unprotected identification attribute with an identification attribute. The identification attribute is intended to contain either a text string or a numeric quantity, such as a random number, which assists the server in locating the shared secret needed to validate the contents of the identityProof attribute. Numeric values MUST be converted to text string representations prior to encoding as UTF8-STRINGs in this attribute. If the identification control attribute is included in

the message, the derivation of the shared secret in step 2 is altered so that the hash of the concatenation of the token and the identity value are hashed rather than just the token.

5.2.1 Hardware Shared Secret Token Generation

The shared secret between the end-entity and the identity verify is sometimes transferred using a hardware device that generates a series of tokens based on some shared secret value. The user can therefore prove their identity by transferring this token in plain text along with a name string. The above protocol can be used with a hardware shared-secret token generation device by the following modifications:

1. The identification attribute MUST be included and MUST contain the hardware-generated token.
2. The shared secret value used above is the same hardware-generated token.
3. All certification requests MUST have a subject name and the subject name MUST contain the fields required to identify the holder of the hardware token device.

5.3 Linking Identity and POP Information

In a PKI Full Request message identity information about the creator/author of the message is carried in the signature of the CMS SignedData object containing all of the certificate requests. Proof-of-possession information for key pairs requesting certification, however, is carried separately for each PKCS#10 or CRMF message. (For keys capable of generating a digital signature, the POP is provided by the signature on the PKCS#10 or CRMF request. For encryption-only keys the controls described in Section 5.7 below are used.) In order to prevent substitution-style attacks we must guarantee that the same entity generated both the POP and proof-of-identity information.

This section describes two mechanisms for linking identity and POP information: witness values cryptographically derived from the shared-secret (Section 5.3.1) and shared-secret/subject DN matching (Section 5.3.2). Clients and servers MUST support the witness value technique. Clients and servers MAY support shared-secret/subject DN matching or other bilateral techniques of similar strength. The idea behind both mechanisms is to force the client to sign some data into each certificate request that can be directly associated with the shared-secret; this will defeat attempts to include certificate requests from different entities in a single Full PKI Request message.

5.3.1 Witness values derived from the shared-secret

The first technique for doing identity-POP linking works by forcing the client to include a piece of information cryptographically-derived from the shared-secret token as a signed extension within each certificate request (PKCS#10 or CRMF) message. This technique is useful if null subject DNs are used (because, for example, the server can generate the subject DN for the certificate based only on the shared secret). Processing begins when the client receives the shared-secret token out-of-band from the server. The client then computes the following values:

1. The client generates a random byte-string, R, which SHOULD be at least 512 bits in length.
2. A SHA1 hash of the token is computed.
3. An HMAC-SHA1 value is then computed over the random value produced in Step 1, as described in [HMAC], using the hash of the token from Step 2 as the shared secret.
4. The random value produced in Step 1 is encoded as the value of an idPOPLinkRandom control attribute. This control attribute MUST be included in the Full PKI Request message.
5. The 160-bit HMAC-SHA1 result from Step 3 is encoded as the value of an idPOPLinkWitness extension to the certificate request.
   a. For CRMF, idPOPLinkWitness is included in the controls section of the CertRequest structure.
   b. For PKCS#10, idPOPLinkWitness is included in the attributes section of the CertificationRequest structure.

Upon receipt, servers MUST verify that each certificate request contains a copy of the idPOPLinkWitness and that its value was derived in the specified manner from the shared secret and the random string included in the idPOPLinkRandom control attribute.

5.3.2 Shared-secret/subject DN matching

The second technique for doing identity-POP linking is to link a particular subject distinguished name (subject DN) to the shared-secrets that are distributed out-of-band and to require that clients using the shared-secret to prove identity include that exact subject DN in every certificate request. It is expected that many client-server connections using shared-secret based proof-of-identity will use this mechanism. (It is common not to omit the subject DN information from the certificate request messages.)

When the shared secret is generated and transferred out-of-band to initiate the registration process (Section 5.2), a particular subject DN is also associated with the shared secret and communicated to the client. (The subject DN generated MUST be unique per entity in

accordance with CA policy; a null subject DN cannot be used. A common practice could be to place the identification value as part of the subject DN.) When the client generates the Full PKI Request message, it MUST use these two pieces of information as follows:

1. The client MUST include the specific subject DN that it received along with the shared secret as the subject name in every certificate request (PKCS#10 and/or CRMF) in the Full PKI Request. The subject names in the requests MUST NOT be null.
2. The client MUST include the identityProof control attribute (Section 5.2), derived from the shared secret, in the Full PKI Request.

The server receiving this message MUST (a) validate the identityProof control attribute and then, (b) check that the subject DN included in each certificate request matches that associated with the shared secret. If either of these checks fails the certificate request MUST be rejected.

5.3.3 Renewal and Re-Key Messages

In a renewal or re-key message, the subject DN in (a) the certificate referenced by the CMS SignerInfo object, and (b) all certificate requests within the request message MUST match according to the standard name match rules described in [PKIXCERT].

5.4 Data Return Control Attribute

The data return control attribute allows clients to send arbitrary data (usually some type of internal state information) to the server and to have the data returned as part of the enrollment response message. Data placed in a data return statement is considered to be opaque to the server. The same control is used for both requests and responses. If the data return statement appears in an enrollment message, the server MUST return it as part of the enrollment response message.

In the event that the information in the data return statement needs to be confidential, it is expected that the client would apply some type of encryption to the contained data, but the details of this are outside the scope of this specification.

An example of using this feature is for a client to place an identifier marking the exact source of the private key material. This might be the identifier of a hardware device containing the private key.

5.5 Add Extensions Control Attribute

The Add Extensions control attribute is used by LRAs in order to specify additional extensions that are to be placed on certificates. This attribute uses the following ASN.1 definition:

```
AddExtensions ::= SEQUENCE {
    pkiDataReference            BodyPartID
    certReferences              SEQUENCE OF BodyPartID,
    extensions                  SEQUENCE OF Extension
}

 -- pkiDataReference field contains the body part id of the
 embedded request message.

 -- certReferences field is a list of references to one or more of
 the payloads contained within a PKIData.  Each element of the
 certReferences sequence MUST be equal to either the bodyPartID of
 a TaggedCertificationRequest or the certReqId of the CertRequest
 within a CertReqMsg.   By definition, the listed extensions are to
 be applied to every element referenced in the certReferences
 sequence.  If a request corresponding to bodyPartID cannot be
 found, the error badRequest is returned referencing this control
 attribute.

 -- extensions field contains the sequence of extensions to be
 applied to the referenced certificate requests.
```

Servers MUST be able to process all extensions defined in [PKIXCERT]. Servers are not required to be able to process every V3 X.509 extension transmitted using this protocol, nor are they required to be able to process other, private extensions. Servers are not required to put all LRA-requested extensions into a certificate. Servers are permitted to modify LRA-requested extensions. Servers MUST NOT alter an extension so as to reverse the meaning of a client-requested extension If a certification request is denied due to the inability to handle a requested extension and a response is returned, the server MUST return a failInfo attribute with the value of unsupportedExt.

If multiple Add Extensions statements exist in an enrollment message, the exact behavior is left up to the certificate issuer policy. However it is recommended that the following policy be used. These rules would be applied to individual extensions within an Add Extensions control attribute (as opposed to an "all or nothing" approach).

1. If the conflict is within a single PKIData object, the certificate request would be rejected with an error of badRequest.

2. If the conflict is between different PKIData objects, the outermost version of the extension would be used (allowing an LRA to override the extension requested by the end-entyt).

5.6 Transaction Management Control Attributes

Transactions are identified and tracked using a transaction identifier. If used, clients generate transaction identifiers and retain their value until the server responds with a message that completes the transaction. Servers correspondingly include received transaction identifiers in the response.

The transactionId attribute identifies a given transaction. It is used between client and server to manage the state of an operation. Clients MAY include a transactionID attribute in request messages. If the original request contains a transactionID attribute, all subsequent request and response messages MUST include the same transactionID attribute. A server MUST use only transactionIds in the outermost PKIdata object. TransactionIds on inner PKIdata objects are for intermediate entities.

Replay protection can be supported through the use of sender and recipient nonces. If nonces are used, in the first message of a transaction, no recipientNonce is transmitted; a senderNonce is instantiated by the message originator and retained for later reference. The recipient of a sender nonce reflects this value back to the originator as a recipientNonce and includes it's own senderNonce. Upon receipt by the transaction originator of this message, the originator compares the value of recipientNonce to its retained value. If the values match, the message can be accepted for further security processing. The received value for senderNonce is also retained for inclusion in the next message associated with the same transaction.

The senderNonce and recipientNonce attribute can be used to provide application-level replay prevention. Clients MAY include a senderNonce in the initial request message. Originating messages include only a value for senderNonce. If a message includes a senderNonce, the response MUST include the transmitted value of the previously received senderNonce as recipientNonce and include new value for senderNonce. A server MUST use only nonces in the outermost PKIdata object. Nonces on inner PKIdata objects are for intermediate entities.

5.7 Proof-of-possession (POP) for encryption-only keys

Everything described in this section is optional to implement, for both servers and clients. Servers MAY require this POP method be used only if another POP method is unavailable. Servers SHOULD reject all requests contained within a PKIData if any required POP is missing for any element within the PKIData.

Many servers require proof that an entity requesting a certificate for a public key actually possesses the corresponding private component of the key pair. For keys that can be used as signature keys, signing the certification request with the private key serves as a POP on that key pair. With keys that can only be used for encryption operations, POP MUST be performed by forcing the client to decrypt a value. See Section 5 of [CRMF] for a detailed discussion of POP.

By necessity, POP for encryption-only keys cannot be done in one round-trip, since there are four distinct phases:

1. Client tells the server about the public component of a new encryption key pair.
2. Server sends the client a POP challenge, encrypted with the presented public encryption key, which the client must decrypt.
3. Client decrypts the POP challenge and sends it back to the server.
4. Server validates the decrypted POP challenge and continues processing the certificate request.

CMC defines two different attributes. The first deals with the encrypted challenge sent from the server to the user in step 2. The second deals with the decrypted challenge sent from the client to the server in step 3.

The encryptedPOP attribute is used to send the encrypted challenge from the server to the client. As such, it is encoded as a tagged attribute within the controlSequence of a ResponseBody. (Note that we assume that the message sent in Step 1 above is an enrollment request and that the response in step 2 is a Full Enrollment Response including a failureInfo specifying that a POP is explicitly required, and providing the POP challenge in the encryptedPOP attribute.)

```
   EncryptedPOP ::= SEQUENCE {

        request        TaggedRequest,
        cms            contentInfo,
        thePOPAlgID    AlgorithmIdentifier,
        witnessAlgID   AlgorithmIdentifier,
        witness        OCTET STRING
```

```
}

DecryptedPOP ::= SEQUENCE {
     bodyPartID     BodyPartID,
     thePOPAlgID    AlgorithmIdentifier,
     thePOP         OCTET STRING
}
```

The encrypted POP algorithm works as follows:

1. The server generates a random value y and associates it with the request.
2. The server returns the encrypted pop with the following fields set:
   a. request is the certificate request in the original request message (it is included here so the client need not key a copy of the request),
   b. cms is an EnvelopedData object, the content type being id-data and the content being the value y. If the certificate request contains a subject key identifier (SKI) extension, then the recipient identifier SHOULD be the SKI. If the issuerAndSerialNumber form is used, the IsserName MUST be encoded as NULL and the SerialNumber as the bodyPartId of the certificate request,
   c. thePOPAlgID contains the algorithm to be used in computing the return POP value,
   d. witnessAlgID contains the hash algorithm used on y to create the field witness,
   e. witness contains the hashed value of y.
3. The client decrypts the cms field to obtain the value y. The client computes H(y) using the witnessAlgID and compares to the value of witness. If the values do not compare or the decryption is not successful, the client MUST abort the enrollment process. The client aborts the process by sending a request message containing a CMCStatusInfo control attribute with failInfo value of popFailed.
4. The client creates the decryptedPOP as part of a new PKIData message. The fields in the decryptedPOP are:
   a. bodyPartID refers to the certificate request in the new enrollment message,
   b. thePOPAlgID is copied from the encryptedPOP,
   c. thePOP contains the possession proof. This value is computed by thePOPAlgID using the value y and request referenced in (4a).
5. The server then re-computes the value of thePOP from its cached value of y and the request and compares to the value of thePOP. If the values do not match, the server MUST NOT issue the certificate. The server MAY re-issue a new challenge or MAY fail

the request altogether.

When defining the algorithms for thePOPAlgID and witnessAlgID care must be taken to ensure that the result of witnessAlgID is not a useful value to shortcut the computation with thePOPAlgID. Clients MUST implement SHA-1 for witnessAlgID. Clients MUST implement HMAC-SHA1 for thePOPAlgID. The value of y is used as the secret value in the HMAC algorithm and the request referenced in (4a) is used as the data. If y is greater than 64 bytes, only the first 64 bytes of y are used as the secret.

One potential problem with the algorithm above is the amount of state that a CA needs to keep in order to verify the returned POP value. This describes one of many possible ways of addressing the problem by reducing the amount of state kept on the CA to a single (or small set) of values.

1. Server generates random seed x, constant across all requests. (The value of x would normally be altered on a regular basis and kept for a short time afterwards.)
2. For certificate request R, server computes y = F(x,R). F can be, for example, HMAC-SHA1(x,R). All that's important for statelessness is that y be consistently computable with only known state constant x and function F, other inputs coming from the cert request structure. y should not be predictable based on knowledge of R, thus the use of a OWF like HMAC-SHA1.

5.8 LRA POP Witnesses Control Attribute

In an enrollment scenario involving an LRAs the CA may allow (or require) the LRA to perform the POP protocol with the entity requesting certification. In this case the LRA needs a way to inform the CA it has done the POP. This control attribute has been created to address this issue.

The ASN.1 structure for the LRA POP witness is as follows:

```
LraPopWitness ::= SEQUENCE {
    pkiDataBodyid   BodyPartID,
    bodyIds         SEQUENCE of BodyPartID
}

-- pkiDataBodyid field contains the body part id of the nested CMS
body object containing the client's full request message.
pkiDataBodyid is set to 0 if the request is in the current
PKIRequest body.
```

```
-- bodyIds contains a list of certificate requests for which the
LRA has performed an out-of-band authentication.  The method of
authentication could be archival of private key material,
challenge-response or other means.
```

If a certificate server does not allow for an LRA to do the POP verification, it returns an error of POPFAILURE. The CA MUST NOT start a challenge-response to re-verify the POP itself.

5.9 Get Certificate Control Attribute

Everything described in this section is optional to implement.

The get certificate control attribute is used to retrieve previously issued certificates from a repository of certificates. A Certificate Authority, an LRA or an independent service may provide this repository. The clients expected to use this facility are those operating in a resource-constrained environment. (An example of a resource-constrained client would be a low-end IP router that does not retain its own certificate in non-volatile memory.)

The get certificate control attribute has the following ASN.1 structure:

```
GetCert ::= SEQUENCE {
    issuerName    GeneralName,
    serialNumber  INTEGER }
```

The service responding to the request will place the requested certificate in the certificates field of a SignedData object. If the get certificate attribute is the only control in a Full PKI Request message, the response would be a Simple Enrollment Response.

5.10 Get CRL Control Attribute

Everything described in this section is optional to implement.

The get CRL control attribute is used to retrieve CRLs from a repository of CRLs. A Certification Authority, an LRA or an independent service may provide this repository. The clients expected to use this facility are those where a fully deployed directory is either infeasible or undesirable.

The get CRL control attribute has the following ASN.1 structure:

```
GetCRL ::= SEQUENCE {
    issuerName    Name,
    cRLName       GeneralName OPTIONAL,
    time          GeneralizedTime OPTIONAL,
    reasons       ReasonFlags OPTIONAL }
```

The fields in a GetCRL have the following meanings:

-- issuerName is the name of the CRL issuer.

-- cRLName may be the value of CRLDistributionPoints in the subject certificate or equivalent value in the event the certificate does not contain such a value.

-- time is used by the client to specify from among potentially several issues of CRL that one whose thisUpdate value is less than but nearest to the specified time. In the absence of a time component, the CA always returns with the most recent CRL.

-- reasons is used to specify from among CRLs partitioned by revocation reason. Implementers should bear in mind that while a specific revocation request has a single CRLReason code--and consequently entries in the CRL would have a single CRLReason code value--a single CRL can aggregate information for one or more reasonFlags.

A service responding to the request will place the requested CRL in the crls field of a SignedData object. If the get CRL attribute is the only control in a full enrollment message, the response would be a simple enrollment response.

5.11 Revocation Request Control Attribute

The revocation request control attribute is used to request that a certificate be revoked.

The revocation request control attribute has the following ASN.1 syntax:

```
RevRequest ::= SEQUENCE {
    issuerName      Name,
    serialNumber    INTEGER,
    reason          CRLReason,
    invalidityDate  GeneralizedTime OPTIONAL,
    sharedSecret    OCTET STRING OPTIONAL,
    comment         UTF8string OPTIONAL }
```

```
   -- issuerName contains the issuerName of the certificate to be
   revoked.

   -- serialNumber contains the serial number of the certificate to
   be revoked

   -- reason contains the suggested CRLReason code for why the
   certificate is being revoked.  The CA can use this value at its
   discretion in building the CRL.

   -- invalidityDate contains the suggested value for the Invalidity
   Date CRL Extension.  The CA can use this value at its discretion
   in building the CRL.

   -- sharedSecret contains a secret value registered by the EE when
   the certificate was obtained to allow for revocation of a
   certificate in the event of key loss.

   -- comment contains a human readable comment.
```

For a revocation request to become a reliable object in the event of a dispute, a strong proof of originator authenticity is required. However, in the instance when an end-entity has lost use of its signature private key, it is impossible for the end-entity to produce a digital signature (prior to the certification of a new signature key pair). The RevRequest provides for the optional transmission from the end-entity to the CA of a shared secret that may be used as an alternative authenticator in the instance of loss of use. The acceptability of this practice is a matter of local security policy.

(Note that in some situations a Registration Authority may be delegated authority to revoke certificates on behalf of some population within its scope control. In these situations the CA would accept the LRA's digital signature on the request to revoke a certificate, independent of whether the end entity still had access to the private component of the key pair.)

Clients MUST provide the capability to produce a digitally signed revocation request control attribute. Clients SHOULD be capable of producing an unsigned revocation request containing the end-entity's shared secret. If a client provides shared secret based self-revocation, the client MUST be capable of producing a revocation request containing the shared secret. Servers MUST be capable of accepting both forms of revocation requests.

The structure of an unsigned, shared secret based revocation request is a matter of local implementation. The shared secret does not need to be encrypted when sent in a revocation request. The shared secret

has a one-time use, that of causing the certificate to be revoked, and public knowledge of the shared secret after the certificate has been revoked is not a problem. Clients need to inform users that the same shared secret SHOULD NOT be used for multiple certificates.

A full response message MUST be returned for a revocation request.

5.12 Registration and Response Information Control Attributes

The regInfo control attribute is for clients and LRAs to pass additional information as part a PKI request. The regInfo control attribute uses the ASN.1 structure:

```
RegInfo ::= OCTET STRING
```

The content of this data is based on bilateral agreement between the client and server.

If a server (or LRA) needs to return information back to a requestor in response to data submitted in a regInfo attribute, then that data is returned as a responseInfo control attribute. The content of the OCTET STRING for response information is based on bilateral agreement between the client and server.

5.13 Query Pending Control Attribute

In some environments, process requirements for manual intervention or other identity checking can cause a delay in returning the certificate related to a certificate request. The query pending attribute allows for a client to query a server about the state of a pending certificate request. The server returns a token as part of the CMCStatusInfo attribute (in the otherInfo field). The client puts the token into the query pending attribute to identify the correct request to the server. The server can also return a suggested time for the client to query for the state of a pending certificate request.

The ASN.1 structure used by the query pending control attribute is:

```
QueryPending ::= OCTET STRING
```

If a server returns a pending state (the transaction is still pending), the otherInfo MAY be omitted. If it is not omitted then the same value MUST be returned (the token MUST NOT change during the request).

5.14 Confirm Certificate Acceptance

Some Certification Authorities require that clients give a positive conformation that the certificates issued to it are acceptable. The Confirm Certificate Acceptance control attribute is used for that purpose. If the CMCStatusInfo on a certificate request is confirmRequired, then the client MUST return a Confirm Certificate

Acceptance prior to any usage of the certificate. Clients SHOULD wait for the response from the server that the conformation has been received.

The confirm certificate acceptance structure is:

```
CMCCertId ::= IssuerSerial

-- CMCCertId contains the issuer and serial number of the
certificate being accepted.
```

Servers MUST return a full enrollment response for a confirm certificate acceptance control.

6. Local Registration Authorities

This specification permits the use of Local Registration Authorities (LRAs). An LRA sits between the end-entity and the Certification Authority. From the end-entity's perspective, the LRA appears to be the Certification Authority and from the server the LRA appears to be a client. LRAs receive the enrollment messages, perform local processing and then forward onto Certificate Authorities. Some of the types of local processing that an LRA can perform include:

- batching multiple enrollment messages together,
- challenge/response POP proofs,
- addition of private or standardized certificate extensions to all requests,
- archival of private key material,
- routing of requests to different CAs.

When an LRA receives an enrollment message it has three options: it may forward the message without modification, it may add a new wrapping layer to the message, or it may remove one or more existing layers and add a new wrapping layer.

When an LRA adds a new wrapping layer to a message it creates a new PKIData object. The new layer contains any control attributes required (for example if the LRA does the POP proof for an encryption key or the addExtension control attribute to modify an enrollment

request) and the client enrollment message. The client enrollment message is placed in the cmsSequence if it is a Full Enrollment message and in the reqSequence if it is a Simple Enrollment message. If an LRA is batching multiple client messages together, then each client enrollment message is placed into the appropriate location in the LRA's PKIData object along with all relevant control attributes.

(If multiple LRAs are in the path between the end-entity and the Certification Authority, this will lead to multiple wrapping layers on the message.)

In processing an enrollment message, an LRA MUST NOT alter any certificate request body (PKCS #10 or CRMF) as any alteration would invalidate the signature on the request and thus the POP for the private key.

An example of how this would look is illustrated by the following figure:

```
   SignedData (by LRA)
     PKIData
       controlSequence
               LRA added control statements
       reqSequence
               Zero or more Simple CertificationRequests from clients
       cmsSequence
               Zero or more Full PKI messages from clients
                  SignedData (by client)
                      PKIData
```

Under some circumstances an LRA is required to remove wrapping layers. The following sections look at the processing required if encryption layers and signing layers need to be removed.

6.1 Encryption Removal

There are two cases that require an LRA to remove or change encryption in an enrollment message. In the first case the encryption was applied for the purposes of protecting the entire enrollment request from unauthorized entities. If the CA does not have a recipient info entry in the encryption layer, the LRA MUST remove the encryption layer. The LRA MAY add a new encryption layer with or without adding a new signing layer.

The second change of encryption that may be required is to change the encryption inside of a signing layer. In this case the LRA MUST remove all signing layers containing the encryption. All control statements MUST be merged according to local policy rules as each

signing layer is removed and the resulting merged controls MUST be placed in a new signing layer provided by the LRA. If the signing layer provided by the end-entity needs to be removed to the LRA can remove the layer.

6.2 Signature Layer Removal

Only two instances exist where an LRA should remove a signature layer on a Full Enrollment message. If an encryption needs to be modified within the message, or if a Certificate Authority will not accept secondary delegation (i.e. multiple LRA signatures). In all other situations LRAs SHOULD NOT remove a signing layer from a message.

If an LRA removes a signing layer from a message, all control statements MUST be merged according to local policy rules. The resulting merged control statements MUST be placed in a new signing layer provided by the LRA.

7. Transport Wrapping

Not all methods of transporting data allow for sending unlabeled raw binary data, in may cases standard methods of encoding can be used to greatly ease this issue. These methods normally consist of wrapping some identification of the content around the binary data, possibly applying an encoding to the data and labeling the data. We document for use three different wrapping methods.

-- MIME wrapping is for transports that are natively MIME based such as HTTP and E-mail.
-- Binary file transport is defined since floppy disk transport is still very common. File transport can be done either as MIME wrapped (section 7.1) or bare (section 7.2).
-- Socket based transport uses the raw BER encoded object.

7.1 MIME Wrapping

MIME wrapping is defined for those environments that are MIME native. These include E-Mail based protocols as well as HTTP.

The basic mime wrapping in this section is taken from [SMIMEV2] and [SMIMEV3]. Simple enrollment requests are encoded using the application/pkcs10 content type. A file name MUST be included either in a content type or content disposition statement. The extension for the file MUST be ".p10".

Simple enrollment response messages MUST be encoded as content-type application/pkcs7-mime. An smime-type parameter MUST be on the content-type statement with a value of "certs-only." A file name with the ".p7c" extension MUST be specified as part of the content-type or content-disposition.

Full enrollment request messages MUST be encoded as content-type application/pkcs7-mime. The smime-type parameter MUST be included with a value of "CMC-enroll". A file name with the ".p7m" extension MUST be specified as part of the content-type or content-disposition statement.

Full enrollment response messages MUST be encoded as content-type application/pkcs7-mime. The smime-type parameter MUST be included with a value of "CMC-response." A file name with the ".p7m" extensions MUST be specified as part of the content-type or content-disposition.

MIME TYPE	File Extension	SMIME-TYPE
application/pkcs10 (simple PKI request)	.p10	N/A
application/pkcs7-mime (full PKI request)	.p7m	CMC-request
application/pkcs7-mime (simple PKI response)	.p7c	certs-only
application/pkcs7-mime (full PKI response)	.p7m	CMC-response

7.2 File-Based Transport

Enrollment messages and responses may also be transferred between clients and servers using file system-based mechanisms, such as when enrollment is performed for an off-line client. When files are used to transport binary, BER-encoded Full Enrollment Request and Response messages, the following file type extensions SHOULD be used:

Message Type	File Extension
Full PKI Request	.crq
Full PKI Response	.crp

7.3 Socket-Based Transport

When enrollment messages and responses are sent over sockets, no wrapping is required. Messages SHOULD be sent in their binary, BER-encoded form.

8. Interoperability

8.1 Mandatory and Optional Algorithms

CMC clients and servers MUST be capable of producing and processing message signatures using the Digital Signature Algorithm [DSA]. DSA signatures MUST be indicated by the DSA AlgorithmIdentifier value (as specified in section 7.2.2 of [PKIXCERT]). PKI clients and servers SHOULD also be capable of producing and processing RSA signatures (as specified in section 7.2.1 of [PKIXCERT]).

CMC clients and servers MUST be capable of protecting and accessing message encryption keys using the Diffie-Hellman (D-H) key exchange algorithm. D-H/3DES protection MUST be indicated by the D-H AlgorithmIdentifier value specified in [CMS]. PKI clients and servers SHOULD also be capable of producing and processing RSA key transport. When used for PKI messages, RSA key transport MUST be indicated as specified in section 7.2.1 of [PKIXCERT].

8.2 Minimum Conformance Requirements

A minimally compliant CMC server:

a) MUST accept a Full PKI Request message
   i) MUST accept CRMF Request Bodies within a Full PKI Request
   ii) MUST accept PKCS#10 Request Bodies within a Full PKI Request
b) MUST accept a Simple Enrollment Request message
c) MUST be able to return a Full PKI Response. (A Full PKI Response is always a valid response, but for interoperability with downlevel clients a compliant server SHOULD use the Simple Enrollment Response whenever possible.)

A minimally-complaint CMC client:

a) MAY use either the Simple Enrollment Message or the Full PKI Request.
   i) clients MUST use PKCS#10 with the Simple Enrollment Message
   ii) clients MAY use either PKCS#10 or CRMF with the Full PKI Request
b) MUST understand the Simple Enrollment Response.
c) MUST understand the Full PKI Response.

9. Security Considerations

Initiation of a secure communications channel between an end-entity and a CA or LRA (and, similarly, between an LRA and another LRA or CA) necessarily requires an out-of-band trust initiation mechanism. For example, a secure channel may be constructed between the end-entity and the CA via IPSEC or TLS. Many such schemes exist and the choice of any particular scheme for trust initiation is outside the scope of this document. Implementers of this protocol are strongly encouraged to consider generally accepted principles of secure key management when integrating this capability within an overall security architecture.

Mechanisms for thwarting replay attacks may be required in particular implementations of this protocol depending on the operational environment. In cases where the CA maintains significant state information, replay attacks may be detectable without the inclusion of the optional nonce mechanisms. Implementers of this protocol need to carefully consider environmental conditions before choosing whether or not to implement the senderNonce and recipientNonce attributes described in section 5.6. Developers of state-constrained PKI clients are strongly encouraged to incorporate the use of these attributes.

Under no circumstances should a signing key be archived. Doing so allows the archiving entity to potentially use the key for forging signatures.

Due care must be taken prior to archiving keys. Once a key is given to an archiving entity, the archiving entity could use the keys in a way not conducive to the archiving entity. Users should be made especially aware that proper verification is made of the certificate used to encrypt the private key material.

Clients and servers need to do some checks on cryptographic parameters prior to issuing certificates to make sure that weak parameters are not used. A description of the small subgroup attack is provided in [X942]. CMC implementations ought to be aware of this attack when doing parameter validations.

10. Acknowledgments

The authors would like to thank Brian LaMacchia for his work in developing and writing up many of the concepts presented in this document. The authors would also like to thank Alex Deacon and Barb Fox for their contributions.

11. References

[CMS] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[CRMF] Myers, M., Adams, C., Solo, D. and D. Kemp, "Internet X.509 Certificate Request Message Format", RFC 2511, March 1999.

[DH] B. Kaliski, "PKCS 3: Diffie-Hellman Key Agreement v1.4"

[DH-POP] H. Prafullchandra, J. Schaad, "Diffie-Hellman Proof-of-Possession Algorithms", Work in Progress.

[HMAC] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[PKCS1] Kaliski, B., "PKCS #1: RSA Encryption, Version 1.5", RFC 2313, March 1998.

[PKCS7] Kaliski, B., "PKCS #7: Cryptographic Message Syntax v1.5", RFC 2315, October 1997.

[PKCS8] RSA Laboratories, "PKCS#8: Private-Key Information Syntax Standard, Version 1.2", November 1, 1993.

[PKCS10] Kaliski, B., "PKCS #10: Certification Request Syntax v1.5", RFC 2314, October 1997.

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

[RFC 2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[SMIMEV2] Dusse, S., Hoffman, P., Ramsdell, B., Lundblade, L. and L. Repka, "S/MIME Version 2 Message Specification", RFC 2311, March 1998.

[SMIMEV3] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[X942] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, June 1999.

12. Authors' Addresses

Michael Myers
VeriSign Inc.
1350 Charleston Road
Mountain View, CA, 94043

Phone: (650) 429-3402
EMail: mmyers@verisign.com

Xiaoyi Liu
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134

Phone: (480) 526-7430
EMail: xliu@cisco.com

Jim Schaad

EMail: jimsch@nwlink.com

Jeff Weinstein

EMail: jsw@meer.net

Appendix A ASN.1 Module

```
EnrollmentMessageSyntax
   { iso(1) identified-organization(3) dod(4) internet(1)
   security(5) mechansims(5) pkix(7) id-mod(0) id-mod-cmc(6) }

   DEFINITIONS IMPLICIT TAGS ::=
   BEGIN

   -- EXPORTS All --
   -- The types and values defined in this module are exported for use
   -- in the other ASN.1 modules.  Other applications may use them for
   -- their own purposes.

   IMPORTS

     -- Information Directory Framework (X.501)
           Name
              FROM InformationFramework { joint-iso-itu-t ds(5)
                   modules(1) informationFramework(1) 3 }

     -- Directory Authentication Framework (X.509)
           AlgorithmIdentifier, AttributeCertificate, Certificate,
           CertificateList, CertificateSerialNumber
              FROM AuthenticationFramework { joint-iso-itu-t ds(5)
                   module(1) authenticationFramework(7) 3 }

     -- PKIX Part 1 - Implicit
        GeneralName, CRLReason, ReasonFlags
        FROM PKIX1Implicit88 {iso(1) identified-organization(3) dod(6)
                internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
                id-pkix1-implicit-88(2)}

     -- PKIX Part 1 - Explicit
        SubjectPublicKeyInfo, Extension
        FROM PKIX1Explicit88 {iso(1) identified-organization(3) dod(6)
                internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
                id-pkix1-explicit-88(1)}

     -- Cryptographic Message Syntax
        ContentInfo, Attribute
          FROM CryptographicMessageSyntax { 1 2 840 113549 1 9 16 0 1}

     -- CRMF
        CertReqMsg
        FROM CRMF { 1 3 6 1 5 5 7 0 5 };

    id-pkix OBJECT IDENTIFIER  ::= { iso(1) identified-organization(3)
```

```
        dod(6) internet(1) security(5) mechanisms(5) pkix(7) }

        id-cmc OBJECT IDENTIFIER ::= {id-pkix 7}   -- CMC controls
        id-cct OBJECT IDENTIFIER ::= {id-pkix 12}  -- CMC content types

    -- The following controls have simple type content (usually OCTET
STRING)

    id-cmc-identification OBJECT IDENTIFIER ::= {id-cmc 2}
    id-cmc-identityProof OBJECT IDENTIFIER ::= {id-cmc 3}
    id-cmc-dataReturn OBJECT IDENTIFIER ::= {id-cmc 4}
    id-cmc-transactionId OBJECT IDENTIFIER ::= {id-cmc 5}
    id-cmc-senderNonce OBJECT IDENTIFIER ::= {id-cmc 6}
    id-cmc-recipientNonce OBJECT IDENTIFIER ::= {id-cmc 7}
    id-cmc-regInfo OBJECT IDENTIFIER ::= {id-cmc 18}
    id-cmc-responseInfo OBJECT IDENTIFIER ::= {id-cmc 19}
    id-cmc-queryPending OBJECT IDENTIFIER ::= {id-cmc 21}
    id-cmc-popLinkRandom OBJECT IDENTIFIER ::= {id-cmc 22)
    id-cmc-popLinkWitness OBJECT IDENTIFIER ::= (id-cmc 23)

    -- This is the content type used for a request message in the
protocol

    id-cct-PKIData OBJECT IDENTIFIER ::= { id-cct 2 }


    PKIData ::= SEQUENCE {
        controlSequence    SEQUENCE SIZE(0..MAX) OF TaggedAttribute,
        reqSequence        SEQUENCE SIZE(0..MAX) OF TaggedRequest,
        cmsSequence        SEQUENCE SIZE(0..MAX) OF TaggedContentInfo,
        otherMsgSequence   SEQUENCE SIZE(0..MAX) OF OtherMsg
    }

    bodyIdMax INTEGER ::= 4294967295

    BodyPartID ::= INTEGER(0..bodyIdMax)

    TaggedAttribute ::= SEQUENCE {
        bodyPartID         BodyPartId,
        attrType           OBJECT IDENTIFIER,
        attrValues         SET OF AttributeValue
    }

    AttributeValue ::= ANY

    TaggedRequest ::= CHOICE {
        tcr               [0] TaggedCertificationRequest,
        crm               [1] CertReqMsg
```

```
    }

    TaggedCertificationRequest ::= SEQUENCE {
        bodyPartID            BodyPartID,
        certificationRequest  CertificationRequest
    }

    CertificationRequest ::= SEQUENCE {
      certificationRequestInfo  SEQUENCE {
        version                   INTEGER,
        subject                   Name,
        subjectPublicKeyInfo      SEQUENCE {
          algorithm                 AlgorithmIdentifier,
          subjectPublicKey          BIT STRING },
        attributes                [0] IMPLICIT SET OF Attribute },
      signatureAlgorithm        AlgorithmIdentifier,
      signature                 BIT STRING
    }

    TaggedContentInfo ::= SEQUENCE {
        bodyPartID              BodyPartId,
        contentInfo             ContentInfo
    }

    OtherMsg ::= SEQUENCE {
        bodyPartID        BodyPartID,
        otherMsgType      OBJECT IDENTIFIER,
        otherMsgValue     ANY DEFINED BY otherMsgType }

    --  This defines the response message in the protocol
    id-cct-PKIResponse OBJECT IDENTIFIER ::= { id-cct 3 }

    ResponseBody ::= SEQUENCE {
        controlSequence   SEQUENCE SIZE(0..MAX) OF TaggedAttribute,
        cmsSequence       SEQUENCE SIZE(0..MAX) OF TaggedContentInfo,
        otherMsgSequence  SEQUENCE SIZE(0..MAX) OF OtherMsg
    }

    -- Used to return status state in a response

    id-cmc-cMCStatusInfo OBJECT IDENTIFIER ::= {id-cmc 1}

    CMCStatusInfo ::= SEQUENCE {
        cMCStatus       CMCStatus,
        bodyList        SEQUENCE SIZE (1..MAX) OF INTEGER,
        statusString    UTF8String OPTIONAL,
        otherInfo        CHOICE {
          failInfo         CMCFailInfo,
```

```
          pendInfo         PendInfo } OPTIONAL
    }

    PendInfo ::= SEQUENCE {
        pendToken        INTEGER,
        pendTime         GENERALIZEDTIME
    }

    CMCStatus ::= INTEGER {
        success         (0),
        -- you got exactly what you asked for
        failed          (2),
        -- you don't get it, more information elsewhere in the message
        pending         (3),
        -- the request body part has not yet been processed,
        -- requester is responsible to poll back on this
        noSupport       (4)
        -- the requested operation is not supported
    }

    CMCFailInfo ::= INTEGER {
        badAlg          (0),
        -- Unrecognized or unsupported algorithm
        badMessageCheck (1),
        -- integrity check failed
        badRequest      (2),
        -- transaction not permitted or supported
        badTime         (3),
        -- Message time field was not sufficiently close to the system
time
        badCertId       (4),
        -- No certificate could be identified matching the provided
criteria
        unsuportedExt   (5),
        -- A requested X.509 extension is not supported by the recipient
CA.
        mustArchiveKeys (6),
        -- Private key material must be supplied
        badIdentity     (7),
        -- Identification Attribute failed to verify
        popRequired     (8),
        -- Server requires a POP proof before issuing certificate
        popFailed       (9),
        -- Server failed to get an acceptable POP for the request
        noKeyReuse      (10)
        -- Server policy does not allow key re-use
        internalCAError (11)
        tryLater        (12)
```

```
    }

    -- Used for LRAs to add extensions to certificate requests
    id-cmc-addExtensions OBJECT IDENTIFIER ::= {id-cmc 8}

    AddExtensions ::= SEQUENCE {
        pkiDataReference    BodyPartID,
        certReferences      SEQUENCE OF BodyPartID,
        extensions          SEQUENCE OF Extension
    }


    id-cmc-encryptedPOP OBJECT IDENTIFIER ::= {id-cmc 9}
    id-cmc-decryptedPOP OBJECT IDENTIFIER ::= {id-cmc 10}

    EncryptedPOP ::= SEQUENCE {
                request       TaggedRequest,
        cms             ContentInfo,
        thePOPAlgID     AlgorithmIdentifier,
        witnessAlgID    AlgorithmIdentifier,
        witness         OCTET STRING
    }

    DecryptedPOP ::= SEQUENCE {
        bodyPartID      BodyPartID,
        thePOPAlgID     AlgorithmIdentifier,
        thePOP          OCTET STRING
    }

    id-cmc-lraPOPWitness OBJECT IDENTIFIER ::= {id-cmc 11}

    LraPopWitness ::= SEQUENCE {
        pkiDataBodyid   BodyPartID,
        bodyIds         SEQUENCE OF BodyPartID
    }


    --
    id-cmc-getCert OBJECT IDENTIFIER ::= {id-cmc 15}

    GetCert ::= SEQUENCE {
        issuerName      GeneralName,
        serialNumber    INTEGER }


    id-cmc-getCRL OBJECT IDENTIFIER ::= {id-cmc 16}

    GetCRL ::= SEQUENCE {
```

```
        issuerName    Name,
        cRLName       GeneralName OPTIONAL,
        time          GeneralizedTime OPTIONAL,
        reasons       ReasonFlags OPTIONAL }

    id-cmc-revokeRequest OBJECT IDENTIFIER ::= {id-cmc 17}

    RevRequest ::= SEQUENCE {
        issuerName            Name,
        serialNumber          INTEGER,
        reason                CRLReason,
       invalidityDate         GeneralizedTime OPTIONAL,
        passphrase            OCTET STRING OPTIONAL,
        comment               UTF8String OPTIONAL }

   id-cmc-confirmCertAcceptance OBJECT IDENTIFIER ::= {pkix-cmc 24}

   CMCCertId ::= IssuerSerial

   -- The following is used to request V3 extensions be added to a
certificate

   id-ExtensionReq OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840)
        rsadsi(113549) pkcs(1) pkcs-9(9) 14}

   ExtensionReq ::= SEQUENCE OF Extension

   -- The following exists to allow Diffie-Hellman Certificate Requests
Messages to be
   -- well-formed

   id-alg-noSignature OBJECT IDENTIFIER ::= {id-pkix id-alg(6) 2}

   NoSignatureValue ::= OCTET STRING

END
```

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.