PKIX Working Group
Internet Draft

February 2002
Expires: August 2002

J. Schaad
Soaring Hawk Consulting
M. Myers
TraceRoute Security
X. Liu
Cisco
J. Weinstein

Certificate Management Messages over CMS

draft-ietf-pkix-2797-bis-01.txt

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.

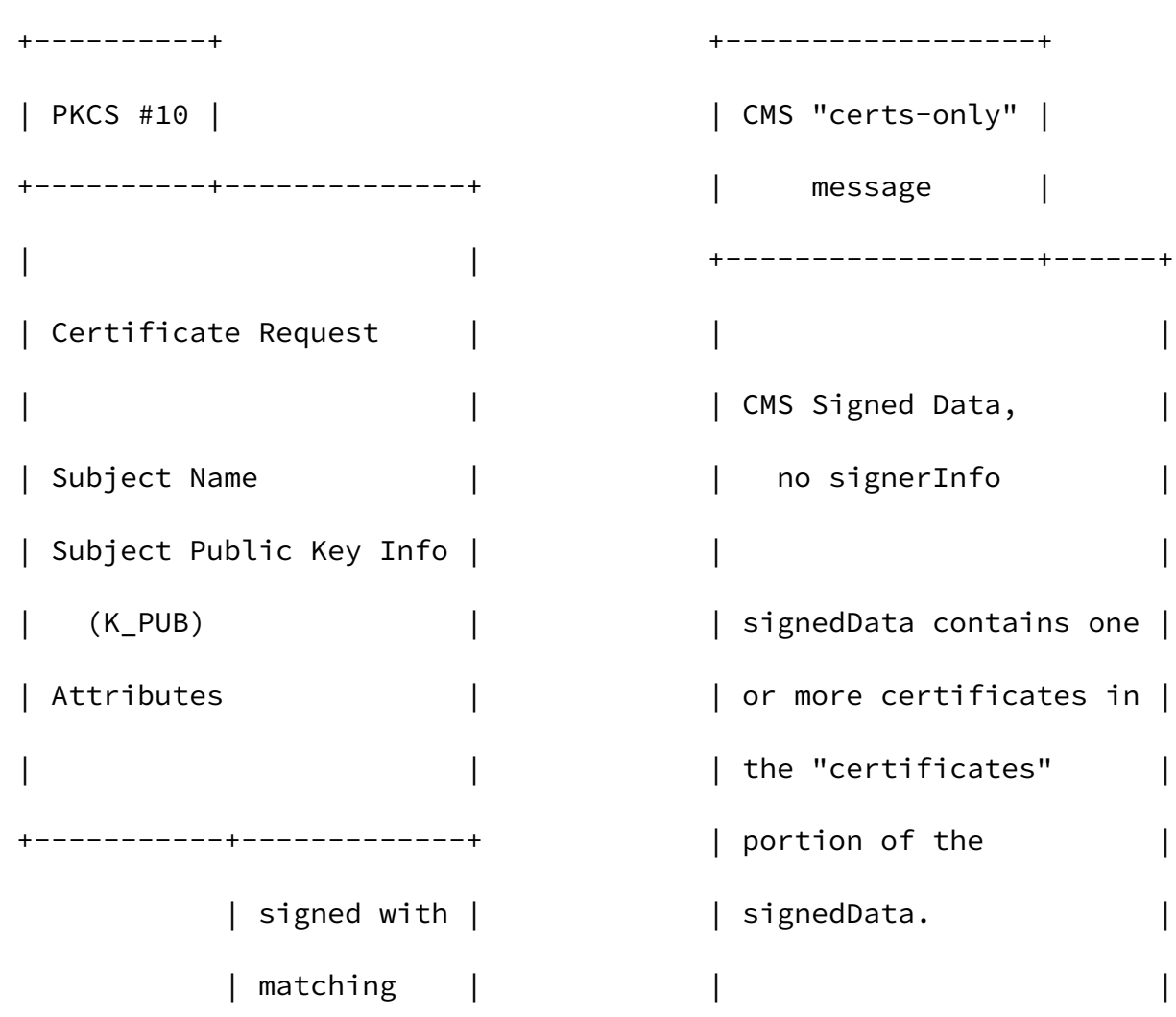
"CMS" refers to the Cryptographic Message Syntax RFC [CMS]. This document provides for basic cryptographic services including encryption and signing with and without key management.

"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.



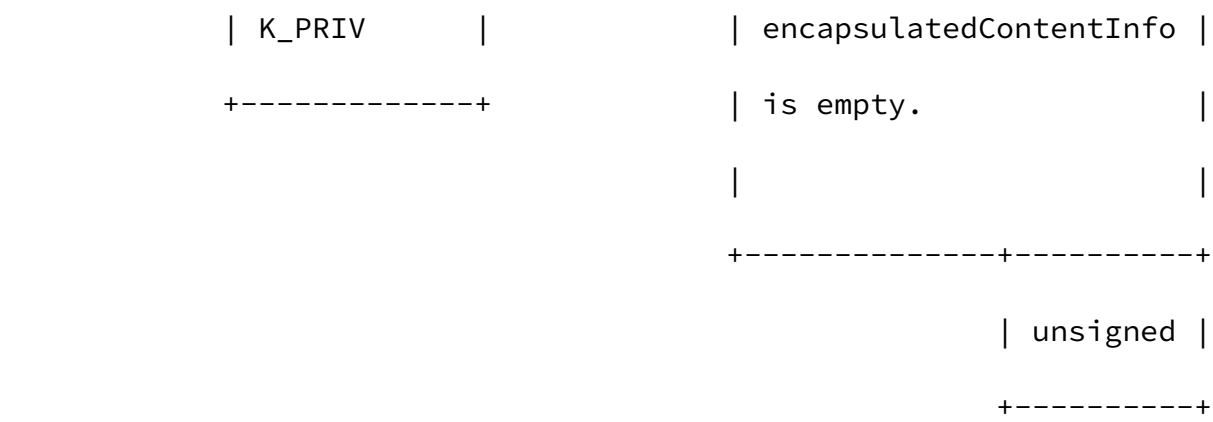
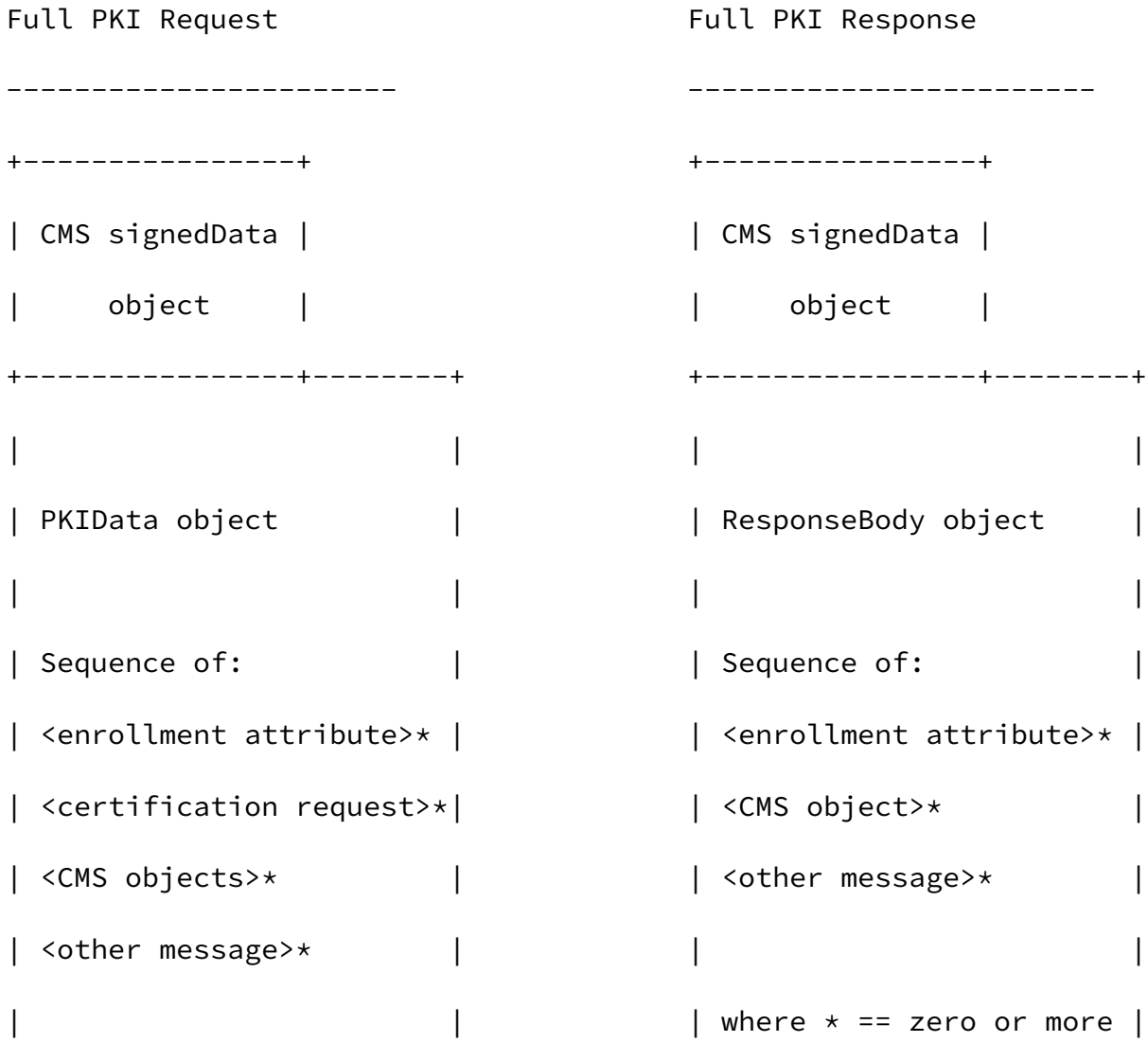


Figure 1: Simple PKI Request and Response Messages



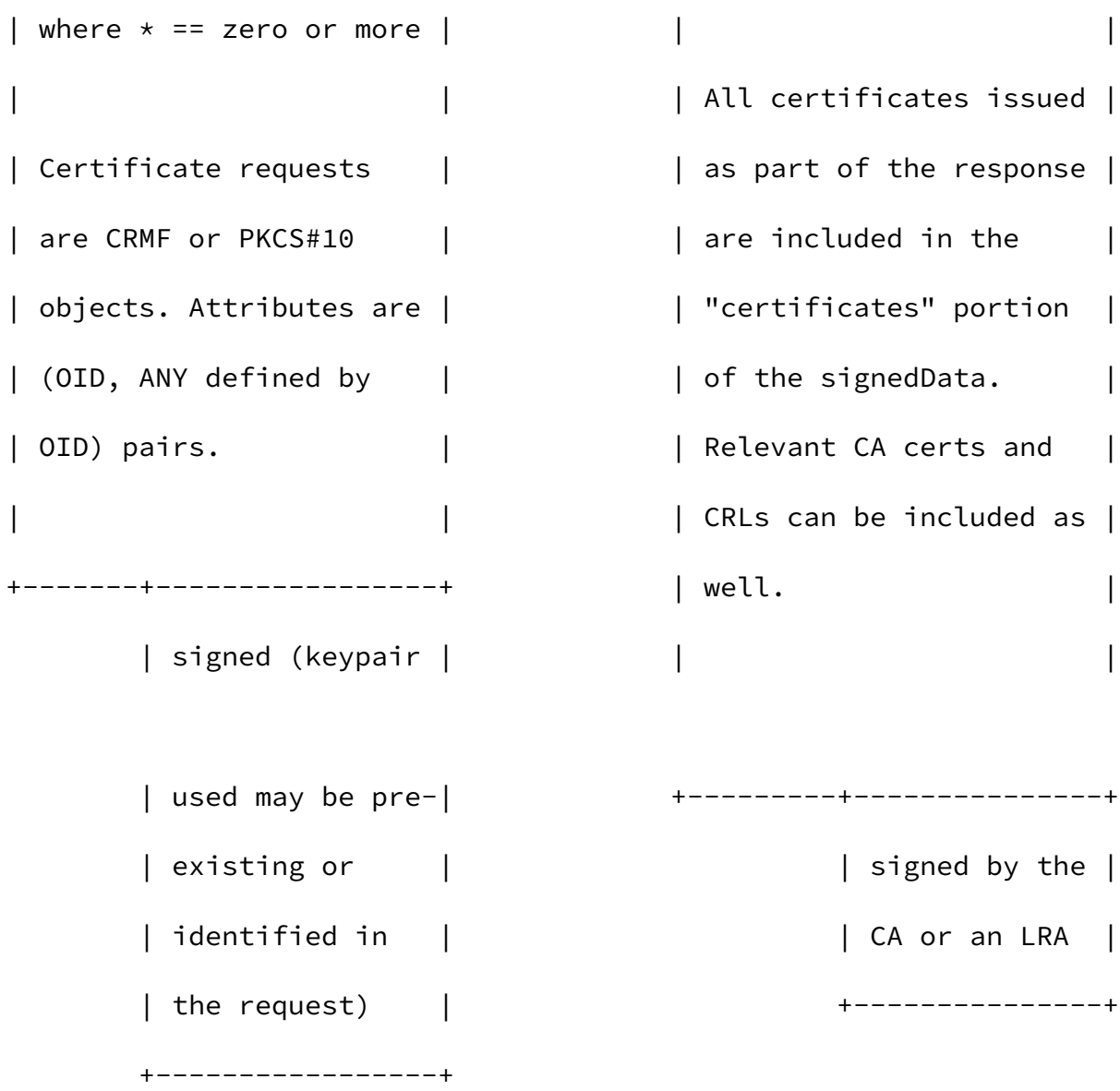


Figure 2: Full PKI Request and Response Messages

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 ::= {id-pkix id-cct(12) 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 requests. The requests can be a CertificateRequest (PKCS10 request), a CertReqMsg or an externally defined request (orm). Details on the first two request types are found in sections 3.3.1 and 3.3.2 respectively. If an externally defined request message is present, but the server does not understand the request (or will not process it), a CMCStatus of noSupport MUST be returned for the request item and no requests processed.

-- cmsSequence consists of a sequence of [CMS] message objects. This protocol uses EnvelopedData, SignedData, EncryptedData and AuthenticatedData. 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.

Processing of this object by a recipient is as follows:

1. All control attributes should be examined and processed in an appropriate manner. The appropriate processing may be either to do

complete processing at this time, ignore the control attribute or to place the control attribute on a to-do list for later processing.

2. An implicit control attribute is then processed for each item in the reqSequence. Again this may be either immediate processing or addition to a to-do list for later processing.

No processing is required for cmsSequence or otherMsgSequence members of the element. If items are present and are not referenced by a control sequence, they are to be ignored.

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 ::= {id-pkix id-cct(12) 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, EncryptedData and AuthenticatedData. 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.

Processing of this object by a recipient is as follows:

1. All control attributes should be examined and processed in an appropriate manner. The appropriate processing may be either to do

complete processing at this time, ignore the control attribute or to place the control attribute on a to-do list for later processing.

2. Additional processing of non-element items includes the saving of certificates and CRLs present in wrapping layers. This type of

processing is based on the consumer of the element and should not be relied on by generators.

No processing is required for cmsSequence or otherMsgSequence members of the element. If items are present and are not referenced by a control sequence, they are to be ignored.

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.

All certificate requests directly encoded into a single PKIData object SHOULD be for the same identity. RAs that batch processing are expected to place the signed PKIData sequences received into the cmsSequence of the PKIData object it generates.

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, but not prohibited, 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.
- When both CRMF and CMC controls exist with equivalent functionality, the CMC control SHOULD be used. The CMC control MUST

override the 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, but not prohibited 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 Discrete Logarithm Signature

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

3.3.3.3 Diffie-Hellman MAC signature

CMC compliant implementations MAY support section 3 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(s).

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.

As part of processing a message the signature(s) MUST be verified.

If the signature does not verify, and the body contains anything other than a status response, a new message containing a status response MUST be returned using a CMCFailInfo with a value of badMessageCheck and a bodyPart of 0.

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. If the decryption on an envelopedData failes, the response is a CMCFailInfo with a value of badMessageCheck and a bodyPart of 0.

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.

3.6.3 Authenticated Data

AuthenticatedData is used for providing origination authentication in those circumstances where a shared-secret exists, but a PKI trust anchor has not yet been established. This is currently only used for the publishAutenticatedData control (section 5.2.16). This control is uses the PKIData body so that new controls with additional policy type information could be included as well.

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.

3.8 Unsigned Attributes

There is sometimes a need to include data in an enrollment message designed to be removed during processing. An example of this is the inclusion of an encrypted private key, where a key archive agent removes the encrypted private key before sending it on to the CA. One side effect of this desire is the fact that every RA which encapsulates this information needs to move the data so that it is not covered by the RA signature. (A client request, encapsulated by an RA cannot have the unsigned attribute removed by the key archive agent without breaking the RA's signature.) This attribute addresses that problem.

This attribute is used to contain the information that is not directly signed by a user. When an RA finds a message that has this attribute in the unsigned or unauthenticated attribute fields of the CMS objects it is aggregating, they are removed from the embedded CMS objects and propagated up to the RA CMS object.

```
id-cmc-UnsignedData OBJECT IDENTIFIER ::= {<TBD>}

CMCUnsignedData ::= SEQUENCE {
```

```
    bodyPartPath        SEQUENCE SIZE (1..MAX) OF BodyPartID,
    identifier          OBJECT IDENTIFIER,
    content             ANY DEFINED BY identifier
}
```

There MUST be at most one CMCUnsignedData attribute in the UnsignedAttribute sequence of a SignerInfo structure. If the attribute appears in one SignerInfo in a sequence, it MUST appear the same in all SignerInfo items and MUST have the same value.

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.

The Simple Enrollment Request message MUST NOT be used if a proof-of-identity needs to be included.

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 object in the PKIData sequence 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,
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.

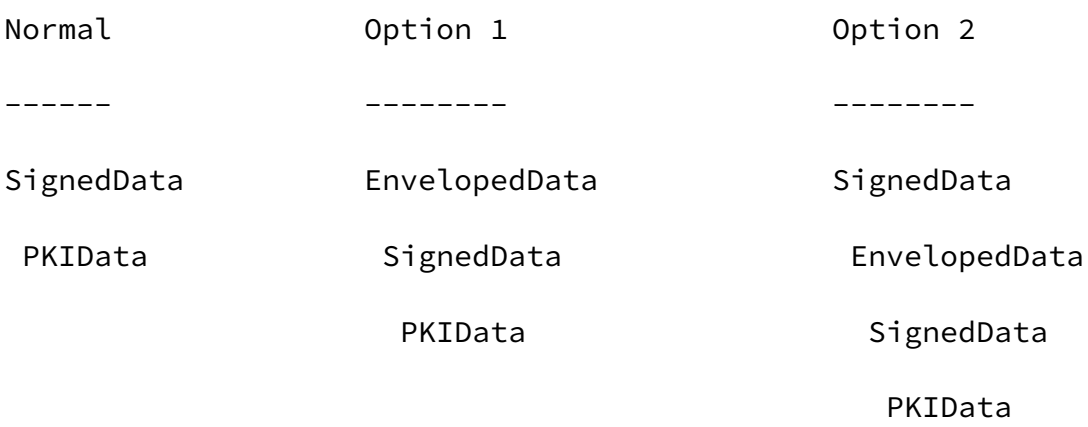
Clients MUST NOT assume the certificates are in any order. Servers SHOULD include all intermediate certificates needed to form complete chains one ore more self-signed certificates, not just the newly issued certificate(s). The server MAY additionally return CRLs in the CRL bag. Servers MAY include the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



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
cmcStatusInfoExt	id-cmc XX	CMCStatusInfoExt
publishTrustRoot	id-cmc XX	CertificateSequence
publishAuthenticatedData	id-cmc XX	AuthPublish
batchRequests	id-cmc XX	BodyPartList
batchResponses	id-cmc XX	BodyPartList

5.1 CMC Status Info Control Attributes

The CMC status info control is used in full PKI Response messages to return information about the processing of a client request. Two controls are described in this section. The first is the preferred control, the second is included for backwards compatibility with RFC 2797.

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. Servers MUST use the CMCStatusInfoExt control, but MAY additionally use the CMCStatusInfo attribute. Clients MUST be able to process the CMCStatusInfoExt control.

5.1.1 Extended CMC Status Info Control Attribute

This control uses the following ASN.1 definition:

```
CMCStatusInfoExt ::= SEQUENCE {
    CMCStatus             CMCStatus,
    BodyList              SEQUENCE SIZE (1..MAX) OF
                                  BodyPartReference,
    StatusString          UTF8String OPTIONAL,
    OtherInfo             CHOICE {
      FailInfo               CMCFailInfo,
      PendInfo               PendInfo,
      ExtendedFailInfo       SEQUENCE {
         FailInfoOID            OBJECT IDENTIFIER,
         FailInfoValue          AttributeValue
      }
    }
}
```

```
   BodyPartReference ::= CHOICE {
      BodyPartID           BodyPartID,
      BodyPartPath         SEQUENCE SIZE (1..MAX) OF BodyPartID
   }

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

   -- cMCStatus is described in section 5.1.3

   -- bodyList contains the list of references to 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 5.1.4. It provides a detailed
error on what the failure was.  This choice is present only if
cMCStatus is failed.
```

-- extendedFailInfo is provided for other users of the enrollment protocol to provided their own error codes. This choice is present only if cMCStatus is failed. Caution should be used in defining new values as they may not be correctly recognized by all clients and servers. The failInfo value of internalCA error may be assumed if the extended error is not recognized.

-- 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.

5.1.2 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
      }
```

-- cMCStatus is described in section 5.1.3

-- 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 5.1.4. It provides a detailed error on what the failure was. This choice is present only if cMCStatus is failed.

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.

5.1.3 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
    }
```

5.1.4 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 shared-secret should be generated in a random manner. 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. Reuse of the shared-secret on enrollment retries makes it easier for the client and to prevent getting out of sync. However, reuse of the shared-secret can potentially open the door for some types of attacks.

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