Limited Additional Mechanisms for PKIX and SMIME
Internet-Draft
Obsoletes: 5272, 6402 (if approved)
Intended status: Standards Track
Expires: 29 November 2025

J. Mandel, Ed
AKAYLA, Inc.
S. Turner, Ed
sn3rd

28 May 2025

Certificate Management over CMS (CMC)
draft-ietf-lamps-rfc5272bis-06

Abstract

This document defines the base syntax for CMC, a Certificate Management protocol using the Cryptographic Message Syntax (CMS). This protocol addresses two immediate needs within the Internet Public Key Infrastructure (PKI) community:

1. The need for an interface to public key certification products and services based on CMS and PKCS #10 (Public Key Cryptography Standard), and

2. The need for a PKI enrollment protocol for encryption-only keys due to algorithm or hardware design.

CMC also requires the use of the transport document and the requirements usage document along with this document for a full definition.

This document obsoletes RFC 5272 and RFC 6402.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-lamps-rfc5272bis/.

Discussion of this document takes place on the Limited Additional Mechanisms for PKIX and SMIME Working Group mailing list (mailto:spasm@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/spasm/. Subscribe at https://www.ietf.org/mailman/listinfo/spasm/.

Source for this draft and an issue tracker can be found at https://github.com/seanturner/cmcbis.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 4

1.1. Protocol Requirements 5
1.2. Requirements Terminology 5
1.3. Changes from RFC 2797 6
1.4. Updates Made by RFC 6402 6
1.5. Changes Since RFC 6402 7
2. Protocol Overview 8
2.1. Terminology 9
2.2. Protocol Requests/Responses 10
3. PKI Requests 12
3.1. Simple PKI Request 12
3.2. Full PKI Request 13
3.2.1. PKIData Content Type 15
3.2.2. Body Part Identification 23
3.2.3. CMC Unsigned Data Attribute 24
4. PKI Responses 25
4.1. Simple PKI Response 25
4.2. Full PKI Response 26
4.2.1. PKIResponse Content Type 26
5. Application of Encryption to a PKI Request/Response 27
6. Controls 28
6.1. CMC Status Info Controls 31
6.1.1. Extended CMC Status Info Control 31
6.1.2. CMC Status Info Control 33
6.1.3. CMCStatus Values 34
6.1.4. CMCFailInfo 35
6.2. Identification and Identity Proof Controls 37
6.2.1. Identity Proof Version 2 Control 37
6.2.2. Identity Proof Control 39
6.2.3. Identification Control 39
6.2.4. Hardware Shared-Secret Token Generation 40
6.3. Linking Identity and POP Information 40
6.3.1. Cryptographic Linkage 41
6.3.2. Shared-Secret/Subject DN Linking 43
6.3.3. Existing Certificate Linking 43
6.4. Data Return Control 44
6.5. RA Certificate Modification Controls 45
6.5.1. Modify Certification Request Control 45
6.5.2. Add Extensions Control 46
6.6. Transaction Identifier Control and Sender and Recipient Nonce Controls 48
6.7. Encrypted and Decrypted POP Controls 49
6.8. RA POP Witness Control 53
6.9. Get Certificate Control 53
6.10. Get CRL Control 54
6.11. Revocation Request Control 55
6.12. Registration and Response Information Controls 57
6.13. Query Pending Control 57
6.14. Confirm Certificate Acceptance Control 58

6.15. Publish Trust Anchors Control 59
6.16. Authenticated Data Control 60
6.17. Batch Request and Response Controls 61
6.18. Publication Information Control 62
6.19. Control Processed Control 63
6.20. RA Identity Proof Witness Control 64
6.21. Response Body Control 65
7. Other Attributes 66
7.1. Change Subject Name Attribute 66
8. Registration Authorities 68
8.1. Encryption Removal 69
8.2. Signature Layer Removal 69
9. Certificate Requirements 70
9.1. Extended Key Usage 70
9.2. Subject Information Access 70
10. Security Considerations 71
11. IANA Considerations 73
12. References 73
12.1. Normative References 73
12.2. Informative References 74
Appendix A. ASN.1 Modules 77
A.1. ASN.1 Module for CMC 77
A.2. ASN.1 Module for PBKDF2 PRFs 90
Appendix B. Enrollment Message Flows 92
B.1. Request of a Signing Certificate 92
B.2. Single Certification Request, But Modified by RA 93
B.3. Direct POP for an RSA or KEM Certificate 96
B.4. Direct POP with No Signature Mechanism 100
Appendix C. Production of Diffie-Hellman Public Key Certification Requests 104
C.1. No-Signature Signature Mechanism 104
Acknowledgments 105
Contributors 105
Authors' Addresses 105

1. Introduction

This document defines the base syntax for CMC, a Certificate Management protocol using the Cryptographic Message Syntax (CMS). 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 PKCS #10, and

2. The need for a PKI enrollment protocol for encryption-only keys due to algorithm or hardware design.

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

This document obsoletes [CMC-PROTv1] and [CMC-Updates].

This document also updates [CMS-ALGS] to add support for additional HMAC algorithms used in the POP Link Witness V2 control.

1.1. Protocol Requirements

The protocol must be based as much as possible on the existing CMS, PKCS #10 [PKCS10] and CRMF (Certificate Request Message Format) [CRMF] specifications.

The protocol must support the current industry practice of a PKCS #10 certification request followed by a PKCS#7 "certs-only" response as a subset of the protocol.

The protocol must easily support the multi-key enrollment protocols required by S/MIME and other groups.

The protocol must supply a way of doing all enrollment operations in a single round trip. When this is not possible the number of round trips is to be minimized.

The protocol must be designed such that all key generation can occur on the client.

Support must exist for the mandatory algorithms used by S/MIME. Support should exist for all other algorithms cited by the S/MIME core documents.

The protocol must contain Proof-of-Possession (POP) methods. Optional provisions for multiple round trip POP will be made if necessary.

The protocol must support deferred and pending responses to enrollment requests for cases where external procedures are required to issue a certificate.

1.2. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.3. Changes from RFC 2797

We have done a major overhaul on the layout of the document. This included two different steps. Firstly we removed some sections from the document and moved them to two other documents. Information on how to transport our messages are now found in [CMC-TRANS]. Information on which controls and sections of this document must be implemented along with which algorithms are required can now be found in [CMC-COMPL].

A number of new controls have been added in this version:

Extended CMC Status Info Section 6.1.1

Publish Trust Anchors Section 6.15

Authenticate Data Section 6.16

Batch Request and Response Processing Section 6.17

Publication Information Section 6.18

Modify Certification Request Section 6.5.1

Control Processed Section 6.19

Identity Proof Section 6.2.2

Identity POP Link Witness V2 Section 6.3.1.1

1.4. Updates Made by RFC 6402

Two new controls have been added:

RA Identity Witness allows for an RA to perform identity checking using the identity and shared-secret, and then tell any following servers that the identity check was successfully performed.

Response Body allows for an RA to identify a nested response for an EE to process.

Created a new attribute, Change Subject Name, that allows a client to request a change in the subject name and subject alternate name fields in a certificate.

Added Extended Key Usages for CMC to distinguish server types.

Defined a new Subject Information Access type to hold locations to contact the CMC server.

Clarified that the use of a pre-existing certificate is not limited to just renewal and rekey messages and is required for support. This formalizes a requirement for the ability to do renewal and rekey that previously was implicit.

1.5. Changes Since RFC 6402

Merged [CMC-Updates] text.

Included the following errata: [erratum8385], [erratum8137], [erratum8027], [erratum7629], [erratum7628], [erratum7627], [erratum7379], [erratum6571], [erratum5931], [erratum4775], [erratum2731], and [erratum2063].

Addressed [erratum3943] for RFC 6402.

To support adopting SHA-256 and HMAC-SHA256, maca-hMAC-SHA256 was added to POPAlgs and mda-sha256 was added to WitnessAlgs. Both were included in the example in Appendix B.

Updated Encrypted and Decrypted POP Controls section to use HMAC-SHA256.

Updated the ASN.1 module to use the 2002 ASN.1 module from [CMC-Updates].

Modified the ASN.1 module for CMC (A.1) to allow the import of ASN.1 Module for PBKDF2 PRFs.

Added a direct POP example to address management of KEM certificates.

Added id-ce-subjectKeyIdentifier to examples.

Clarified that subjectKeyIdentifier choice used with id-alg-noSignature.

Update CMC Control Attribute Table to include raIdentityWitness and responseBody from RFC 6402

2. Protocol Overview

A PKI enrollment transaction in this specification is generally composed of a single round trip of messages. In the simplest case a PKI enrollment request, henceforth referred to as a PKI Request, is sent from the client to the server and a PKI enrollment response, henceforth referred to as a PKI Response, is then returned from the server to the client. In more complicated cases, such as delayed certificate issuance, more than one round trip is required.

This specification defines two PKI Request types and two PKI Response types.

PKI Requests are formed using either the PKCS #10 or CRMF structure. The two PKI Requests are:

- Simple PKI Request: the bare PKCS #10 (in the event that no other services are needed), and

- Full PKI Request: one or more PKCS #10, CRMF or Other Request Messages structures wrapped in a CMS encapsulation as part of a PKIData.

PKI Responses are based on SignedData or AuthenticatedData [CMS]. The two PKI Responses are

- Simple PKI Response: a "certs-only" SignedData (in the event no other services are needed), or

- Full PKI Response: a PKIResponse content type wrapped in a SignedData.

No special services are provided for either renewal (i.e., a new certificate with the same key) or rekey (i.e., a new certificate with a new key) of client certificates. Instead renewal and rekey requests look the same as any certification request, except that the identity proof is supplied by existing certificates from a trusted CA. (This is usually the same CA, but could be a different CA in the same organization where naming is shared.)

No special services are provided to distinguish between a rekey request and a new certification request (generally for a new purpose). A control to unpublish a certificate would normally be included in a rekey request, and be omitted in a new certification request. CAs or other publishing agents are also expected to have policies for removing certificates from publication either based on new certificates being added or the expiration or revocation of a certificate.

A provision exists for RAs to participate in the protocol by taking PKI Requests, wrapping them in a second layer of PKI Request with additional requirements or statements from the RA and then passing this new expanded PKI Request on to the CA.

This specification makes no assumptions about the underlying transport mechanism. The use of CMS does not imply an email-based transport. Several different possible transport methods are defined in [CMC-TRANS].

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

2.1. Terminology

There are several different terms, abbreviations, and acronyms used in this document. These are defined here, in no particular order, 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.

Registration Authority (RA) or Local RA (LRA) refers to an entity that acts as an intermediary between the EE and the CA. Multiple RAs can exist between the end-entity and the Certification Authority. RAs may perform additional services such as key generation or key archival. This document uses the term RA for both RA and LRA.

Certification Authority (CA) refers to the entity that issues certificates.

Client refers to an entity that creates a PKI Request. In this document, both RAs and EEs can be clients.

Server refers to the entities that process PKI Requests and create PKI Responses. In this document, both CAs and RAs can be servers.

PKCS #10 refers to the Public Key Cryptography Standard #10 [PKCS10] which defines a certification request syntax.

CRMF refers to the Certificate Request Message Format RFC [CRMF]. CMC uses this certification request syntax defined in this document as part of the 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.

PKI Request/Response refers to the requests/responses described in this document. PKI Requests include certification requests, revocation requests, etc. PKI Responses include certs-only messages, failure messages, etc.

Proof-of-Identity refers to the client proving they are who they say that they are to the server.

Enrollment or certification request refers to the process of a client requesting a certificate. A certification request is a subset of the PKI Requests.

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. The different types of POP are:

- Signature provides the required POP by a signature operation over some data.
- Direct provides the required POP operation by an encrypted challenge/response mechanism.
- Indirect provides the required POP operation by returning the issued certificate in an encrypted state. (This method is not used by CMC.)
- Publish provides the required POP operation by providing the private key to the certificate issuer. (This method is not currently used by CMC. It would be used by Key Generation or Key Escrow extensions.)
- Attested provides the required POP operation by allowing a trusted entity to assert that the POP has been proven by one of the above methods.

Object IDentifier (OID) is a primitive type in Abstract Syntax Notation One (ASN.1).

2.2. Protocol Requests/Responses

Figure 1 shows the Simple PKI Requests and Responses. The contents of Simple PKI Request and Response are detailed in Section 3.1 and Section 4.1.

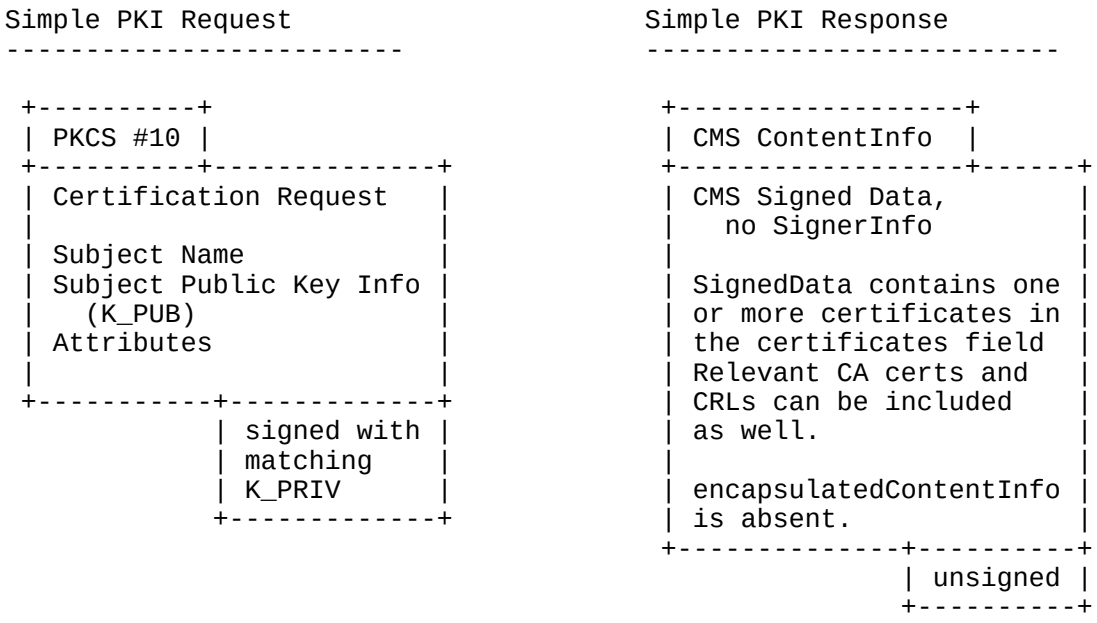


Figure 1: Simple PKI Requests and Responses

Figure 2 shows the Full PKI Requests and Responses. The contents of the Full PKI Request and Response are detailed in Section 3.2 and Section 4.2.

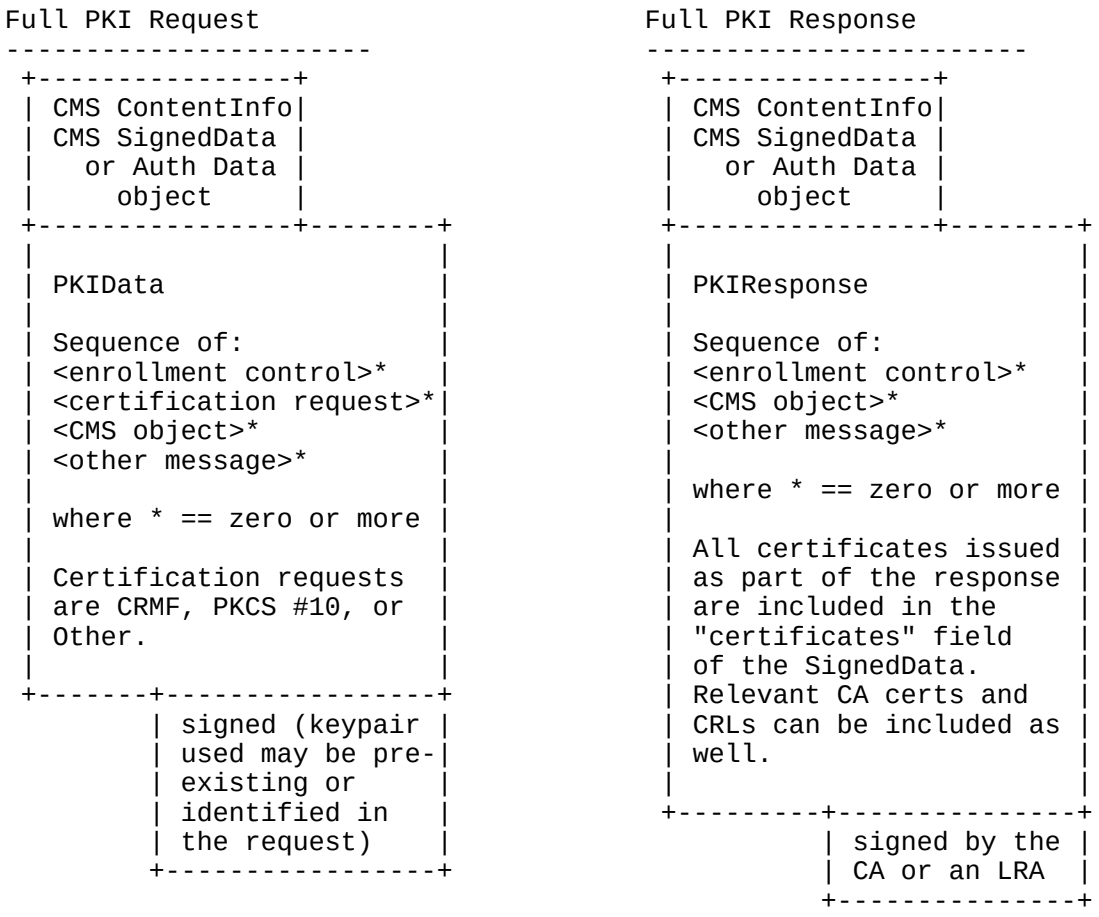


Figure 2: Full PKI Requests and Responses

3. PKI Requests

Two types of PKI Requests exist. This section gives the details for both types.

3.1. Simple PKI Request

A Simple PKI Request uses the PKCS #10 syntax CertificationRequest [PKCS10].

When a server processes a Simple PKI Request, the PKI Response returned is:

Simple PKI Response on success.

Full PKI Response on failure. The server MAY choose not to return a PKI Response in this case.

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

The Simple PKI Request cannot be used if the private key is not capable of producing some type of signature (i.e., Diffie-Hellman (DH) and Elliptic Curve Diffie-Hellman (ECDH) keys can use the signature algorithms in [DH-POP] for production of the signature).

The Simple PKI Request cannot be used for any of the advanced services specified in this document.

The client MAY incorporate one or more X.509v3 extensions in any certification request based on PKCS #10 as an ExtensionReq attribute. The ExtensionReq attribute is defined as:

```
ExtensionReq ::= SEQUENCE SIZE (1..MAX) OF Extension
```

where Extension is imported from [PKIXCERT] and ExtensionReq is identified by:

```
id-ExtensionReq OBJECT IDENTIFIER ::= { iso(1) member-body(2)
  us(840) rsadsi(113549) pkcs(1) pkcs-9(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 X.509v3 extensions transmitted using this protocol, nor are they required to be able to process 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 keyAgreement to digitalSignature.) If a certification request is denied due to the inability to handle a requested extension and a PKI Response is returned, the server MUST return a PKI Response with a CMCFailInfo value with the value unsupportedExt.

3.2. Full PKI Request

The Full PKI Request provides the most functionality and flexibility.

The Full PKI Request is encapsulated in either a SignedData or an AuthenticatedData with an encapsulated content type of ’id-cct-PKIData’ (Section 3.2.1).

When a server processes a Full PKI Request, a PKI Response MUST be returned. The PKI Response returned is:

Simple PKI Response if the enrollment was successful and only certificates are returned. (A CMCStatusInfoV2 control with success is implied.)

Full PKI Response if the enrollment was successful and information is returned in addition to certificates, if the enrollment is pending, or if the enrollment failed.

If SignedData is used, the signature can be generated using either the private key material of an embedded signature certification request (i.e., included in the TaggedRequest tcr or crm fields) or a previously certified signature key. If the private key of a signature certification request is used, then:

1. The certification request containing the corresponding public key MUST include a Subject Key Identifier extension.

2. The subjectKeyIdentifier form of the signerIdentifier in SignerInfo MUST be used.

3. 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 in the SignedData.

If AuthenticatedData is used, then:

1. The Password Recipient Info option of RecipientInfo MUST be used.

2. A randomly generated key is used to compute the Message Authentication Code (MAC) value on the encapsulated content.

3. The input for the key derivation algorithm is a concatenation of the identifier (encoded as UTF8) and the shared-secret.

When creating a PKI Request to renew or rekey a certificate:

1. The Identification and Identity Proof controls are absent. The same information is provided by the use of an existing certificate from a CA when signing the PKI Request. In this case, the CA that issued the original certificate and the CA the request is made to will usually be the same, but could have a common operator.

2. CAs and RAs can impose additional restrictions on the signing certificate used. They may require that the most recently issued signing certificate for a client be used.

3. Some CAs may prevent renewal operations (i.e., reuse of the same keys). In this case the CA MUST return a PKI Response with noKeyReuse as the CMCFailInfo failure code.

3.2.1. PKIData Content Type

The PKIData content type is used for the Full PKI Request. A PKIData content type is identified by:

```
id-cct-PKIData OBJECT IDENTIFIER ::= { id-pkix id-cct(12) 2 }
```

The ASN.1 structure corresponding to the PKIData 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
}
```

The fields in PKIData have the following meaning:

controlSequence is a sequence of controls. The controls defined in this document are found in Section 6. Controls can be defined by other parties. Details on the TaggedAttribute structure can be found in Section 3.2.1.1.

reqSequence is a sequence of certification requests. The certification requests can be a CertificationRequest (PKCS #10), a CertReqMsg (CRMF), or an externally defined PKI request. Full details are found in Section 3.2.1.2. If an externally defined certification request is present, but the server does not understand the certification request (or will not process it), a CMCStatus of noSupport MUST be returned for the certification request item and no other certification requests are processed.

cmsSequence is a sequence of [CMS] message objects. See Section 3.2.1.3 for more details.

otherMsgSequence is a sequence of arbitrary data objects. Data objects placed here are referred to by one or more controls. This allows for controls to use large amounts of data without the data being embedded in the control. See Section 3.2.1.4 for more details.

All certification requests encoded into a single PKIData SHOULD be for the same identity. RAs that batch process (see Section 6.17) are expected to place the PKI Requests received into the cmsSequence of a PKIData.

Processing of the PKIData by a recipient is as follows:

1. All controls should be examined and processed in an appropriate manner. The appropriate processing is to complete processing at this time, to ignore the control, or to place the control on a to-do list for later processing. Controls can be processed in any order; the order in the sequence is not significant.

2. Items in the reqSequence are not referenced by a control. These items, which are certification requests, also need to be processed. As with controls, the appropriate processing can be either immediate processing or addition to a to-do list for later processing.

3. Finally, the to-do list is processed. In many cases, the to-do list will be ordered by grouping specific tasks together.

No processing is required for cmsSequence or otherMsgSequence members of PKIData if they are present and are not referenced by a control. In this case, the cmsSequence and otherMsgSequence members are ignored.

3.2.1.1. Control Syntax

The actions to be performed for a PKI Request/Response are based on the included controls. Each control consists of an object identifier and a value based on the object identifier.

The syntax of a control is:

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

AttributeValue ::= ANY
```

The fields in TaggedAttribute have the following meaning:

bodyPartID is a unique integer that identifies this control.

attrType is the OID that identifies the control.

attrValues is the data values used in processing the control. The structure of the data is dependent on the specific control.

The final server MUST fail the processing of an entire PKIData if any included control is not recognized, that control is not already marked as processed by a Control Processed control (see Section 6.19) and no other error is generated. The PKI Response MUST include a CMCFailInfo value with the value badRequest and the bodyList MUST contain the bodyPartID of the invalid or unrecognized control(s). A server is the final server if and only if it is not passing the PKI Request on to another server. A server is not considered to be the final server if the server would have passed the PKI Request on, but instead it returned a processing error.

The controls defined by this document are found in Section 6.

3.2.1.2. Certification Request Formats

Certification Requests are based on PKCS #10, CRMF, or Other Request formats. Section 3.2.1.2.1 specifies the requirements for clients and servers dealing with PKCS #10. Section 3.2.1.2.2 specifies the requirements for clients and servers dealing with CRMF. Section 3.2.1.2.3 specifies the requirements for clients and servers dealing with Other Request.

```
TaggedRequest ::= CHOICE {
  tcr               [0] TaggedCertificationRequest,
  crm               [1] CertReqMsg,
  orm               [2] SEQUENCE {
    bodyPartID            BodyPartID,
    requestMessageType    OBJECT IDENTIFIER,
    requestMessageValue   ANY DEFINED BY requestMessageType
  }
}
```

The fields in TaggedRequest have the following meaning:

tcr is a certification request that uses the PKCS #10 syntax. Details on PKCS #10 are found in Section 3.2.1.2.1.

crm is a certification request that uses the CRMF syntax. Details on CRMF are found in Section 3.2.1.2.2.

orm is an externally defined certification request. One example is an attribute certification request. The fields of this structure are:

- bodyPartID is the identifier number for this certification request. Details on body part identifiers are found in Section 3.2.2.

- requestMessageType identifies the other request type. These values are defined outside of this document.

- requestMessageValue is the data associated with the other request type.

3.2.1.2.1. PKCS #10 Certification Syntax

A certification request based on PKCS #10 uses the following ASN.1 structure:

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

The fields in TaggedCertificationRequest have the following meaning:

bodyPartID is the identifier number for this certification request. Details on body part identifiers are found in Section 3.2.2.

certificationRequest contains the PKCS-#10-based certification request. Its fields are described in [PKCS10].

When producing a certification request based on PKCS #10, clients MUST produce the certification request with a subject name and public key. Some PKI products are operated using a central repository of information to assign subject names upon receipt of a certification request. To accommodate this mode of operation, the subject field in a CertificationRequest MAY be NULL, but MUST be present. CAs that receive a CertificationRequest with a NULL subject field MAY reject such certification requests. If rejected and a PKI Response is returned, the CA MUST return a PKI Response with the CMCFailInfo value with the value badRequest.

3.2.1.2.2. CRMF Certification Syntax

A CRMF message uses the following ASN.1 structure (defined in [CRMF] and included here for convenience):

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

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

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

The fields in CertReqMsg are explained in [CRMF].

This document imposes the following additional restrictions on the construction and processing of CRMF certification requests:

When a Full PKI Request includes a CRMF certification request, both the subject and publicKey fields in the CertTemplate MUST be defined. The subject field can 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 certification request. Equivalent functionality is provided in the CMC regInfo control (Section 6.12).

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. 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 CRMF certification request. Servers MUST be able to process all extensions defined, but not prohibited in [PKIXCERT]. Servers are not required to be able to process other X.509v3 extensions transmitted using this protocol, nor are they required to be able to process 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 keyAgreement to digitalSignature.) If a certification request is denied due to the inability to handle a requested extension, the server MUST respond with a Full PKI Response with a CMCFailInfo value with the value of unsupportedExt.

3.2.1.2.3. Other Certification Request

This document allows for other certification request formats to be defined and used as well. An example of an other certification request format is one for Attribute Certificates. These other certification request formats are defined by specifying an OID for identification and the structure to contain the data to be passed.

3.2.1.3. Content Info Objects

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

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

The fields in TaggedContentInfo have the following meaning:

bodyPartID is a unique integer that identifies this content info object.

contentInfo is a ContentInfo object (defined in [CMS]).

The four content types used in cmsSequence are AuthenticatedData, Data, EnvelopedData, and SignedData. All of these content types are defined in [CMS].

3.2.1.3.1. Authenticated Data

The AuthenticatedData content type provides a method of doing pre-shared-secret-based validation of data being sent between two parties. Unlike SignedData, it does not specify which party actually generated the information.

AuthenticatedData provides origination authentication in those circumstances where a shared-secret exists, but a PKI-based trust has not yet been established. No PKI-based trust may have been established because a trust anchor has not been installed on the client or no certificate exists for a signing key.

AuthenticatedData content type is used by this document for:

- The id-cmc-authData control (Section 6.16), and
- The top-level wrapper in environments where an encryption-only key is being certified or where a shared-secret exists, but a PKI-based trust (needed for SignedData) has not yet been established.

This content type can include both PKIData and PKIResponse as the encapsulated content types. These embedded content types can contain additional controls that need to be processed.

3.2.1.3.2. Data

The Data content type allows for general transport of unstructured data.

The Data content type is used by this document for:

- Holding the encrypted random value y for POP proof in the encrypted POP control (see Section 6.7).

3.2.1.3.3. Enveloped Data

The EnvelopedData content type provides for shrouding of data.

The EnvelopedData content type is the primary confidentiality method for sensitive information in this protocol. EnvelopedData can provide encryption of an entire PKI Request (see Section 5). EnvelopedData can also be used to wrap private key material for key archival. If the decryption on an EnvelopedData fails, a Full PKI Response is returned with a CMCFailInfo value of badMessageCheck and a bodyPartID of 0.

3.2.1.3.4. Signed Data

The SignedData content type provides for authentication and integrity.

The SignedData content type is used by this document for:

- The outer wrapper for a PKI Request.

- The outer wrapper for a PKI Response.

As part of processing a PKI Request/Response, the signature(s) MUST be verified. If the signature does not verify and the PKI Request/Response contains anything other than a CMC Status Info control, a Full PKI Response containing a CMC Status Info control MUST be returned using a CMCFailInfo with a value of badMessageCheck and a bodyPartID of 0.

For the PKI Response, SignedData allows the server to sign the returning data, if any exists, and to carry the certificates and CRLs corresponding to the PKI Request. If no data is being returned beyond the certificates and CRLs, there is no 'eContent' field in the 'EncapsulatedContentInfo' and no 'SignerInfo'.

Only if the server is unable to sign the response (and unable to use any RecipientInfo options of the AuthenticatedData content type), and at the same time it should send a negative response, Full PKI Response SignedData type containing a CMC Status Info control MUST be returned using a CMCFailInfo with a value of internalCAError and a bodyPartID of 0, and the eContent field in the EncapsulatedContentInfo as well as SignerInfo fields MUST NOT be populated.

3.2.1.4. Other Message Bodies

The otherMsgSequence field of the PKI Request/Response allows for arbitrary data objects to be carried as part of a PKI Request/Response. This is intended to contain a data object that is not already wrapped in a cmsSequence field Section 3.2.1.3. The data object is ignored unless a control references the data object by bodyPartID.

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

The fields in OtherMsg have the following meaning:

bodyPartID is the unique id identifying this data object.

otherMsgType is the OID that defines the type of message body.

otherMsgValue is the data.

3.2.2. Body Part Identification

Each element of a PKIData or PKIResponse has an associated body part identifier. The body part identifier is a 4-octet integer using the ASN.1 of:

```
bodyIdMax INTEGER ::= 4294967295

BodyPartID ::= INTEGER(0..bodyIdMax)
```

Body part identifiers are 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. Body part identifiers can be duplicated in different layers (for example, a PKIData embedded within another).

The bodyPartID value of 0 is reserved for use as the reference to the current PKIData object.

Some controls, such as the Add Extensions control Section 6.5.2, use the body part identifier in the pkiDataReference field to refer to a PKI Request in the current PKIData. Some controls, such as the Extended CMC Status Info control Section 6.1.1, will also use body part identifiers to refer to elements in the previous PKI Request/ Response. This allows an error to be explicit about the control or PKI Request to which the error applies.

A BodyPartList contains a list of body parts in a PKI Request/ Response (i.e., the Batch Request control in Section 6.17). The ASN.1 type BodyPartList is defined as:

```
BodyPartList ::= SEQUENCE SIZE (1..MAX) OF BodyPartID
```

A BodyPartPath contains a path of body part identifiers moving through nesting (i.e., the Modify Certification Request control in Section 6.5.1). The ASN.1 type BodyPartPath is defined as:

```
BodyPartPath ::= SEQUENCE SIZE (1..MAX) OF BodyPartID
```

3.2.3. CMC Unsigned Data Attribute

There is sometimes a need to include data in a PKI Request designed to be removed by an RA 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 that every RA that encapsulates this information needs to move the data so that it is not covered by that RA’s signature. (A client PKI Request encapsulated by an RA cannot have a signed control removed by the Key Archive Agent without breaking the RA’s signature.) The CMC Unsigned Data attribute addresses this problem.

The CMC Unsigned Data attribute contains information that is not directly signed by a client. When an RA encounters this attribute in the unsigned or unauthenticated attribute field of a request it is aggregating, the CMC Unsigned Data attribute is removed from the request prior to placing the request in a cmsSequence and placed in the unsigned or unauthenticated attributes of the RA’s signed or authenticated data wrapper.

The CMC Unsigned Data attribute is identified by:

```
id-aa-cmc-unsignedData OBJECT IDENTIFIER ::= { id-aa 34 }
```

The CMC Unsigned Data attribute has the ASN.1 definition:

```
CMCUnsignedData ::= SEQUENCE {
  bodyPartPath        BodyPartPath,
  identifier          OBJECT IDENTIFIER,
  content             ANY DEFINED BY identifier
}
```

The fields in CMCUnsignedData have the following meaning:

bodyPartPath is the path pointing to the control associated with this data. When an RA moves the control in an unsigned or unauthenticated attribute up one level as part of wrapping the data in a new SignedData or AuthenticatedData, the body part identifier of the embedded item in the PKIData is prepended to the bodyPartPath sequence.

identifier is the OID that defines the associated data.

content is the data.

There MUST be at most one CMC Unsigned Data attribute in the UnsignedAttribute sequence of a SignerInfo or in the UnauthenticatedAttribute sequence of an AuthenticatedData. UnsignedAttribute consists of a set of values; the attribute can have any number of values greater than zero in that set. If the CMC Unsigned Data attribute is in one SignerInfo or AuthenticatedData, it MUST appear with the same values(s) in all SignerInfo and AuthenticatedData items.

4. PKI Responses

Two types of PKI Responses exist. This section gives the details on both types.

4.1. Simple PKI Response

Clients MUST be able to process the Simple PKI Response. The Simple PKI Response consists of a SignedData with no EncapsulatedContentInfo and no SignerInfo. The certificates requested in the PKI Response are returned in the certificate field of the SignedData.

Clients MUST NOT assume the certificates are in any order. Servers SHOULD include all intermediate certificates needed to form complete certification paths to one or more trust anchors, 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 use the certificate as a trust anchor. (The Publish Trust Anchors control (Section 6.15) should be used in the event that the server intends the client to accept one or more certificates as trust anchors. This requires the use of the Full PKI Response message.)

4.2. Full PKI Response

Clients MUST be able to process a Full PKI Response.

The Full PKI Response consists of a SignedData or AuthenticatedData encapsulating a PKIResponse content type. The certificates issued in a PKI Response are returned in the certificates field of the immediately encapsulating SignedData.

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 trust anchors, not just the newly issued certificate(s). The server MAY additionally return CRLs in the CRL bag. Servers MAY include 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 MAY provide a mechanism to enable the user to explicitly use the certificate as a trust anchor. (The Publish Trust Anchors control (Section 6.15) exists for the purpose of allowing for distribution of trust anchor certificates. If a trusted anchor publishes a new trusted anchor, this is one case where automated trust of the new trust anchor could be allowed.)

4.2.1. PKIResponse Content Type

The PKIResponse content type is used for the Full PKI Response. The PKIResponse content type is identified by:

```
id-cct-PKIResponse OBJECT IDENTIFIER ::= { id-pkix id-cct(12) 3  }
```

The ASN.1 structure corresponding to the PKIResponse content type is:

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

  ReponseBody ::= PKIResponse
```

Note: In [RFC2797], this ASN.1 type was named ResponseBody. It has been renamed to PKIResponse for clarity and the old name kept as a synonym.

The fields in PKIResponse have the following meaning:

controlSequence is a sequence of controls. The controls defined in this document are found in Section 6. Controls can be defined by other parties. Details on the TaggedAttribute structure are found in Section 3.2.1.1.

cmsSequence is a sequence of [CMS] message objects. See Section 3.2.1.3 for more details.

otherMsgSequence is a sequence of arbitrary data objects. Data objects placed here are referred to by one or more controls. This allows for controls to use large amounts of data without the data being embedded in the control. See Section 3.2.1.4 for more details.

Processing of PKIResponse by a recipient is as follows:

1. All controls should be examined and processed in an appropriate manner. The appropriate processing is to complete processing at this time, to ignore the control, or to place the control on a to-do list for later processing.

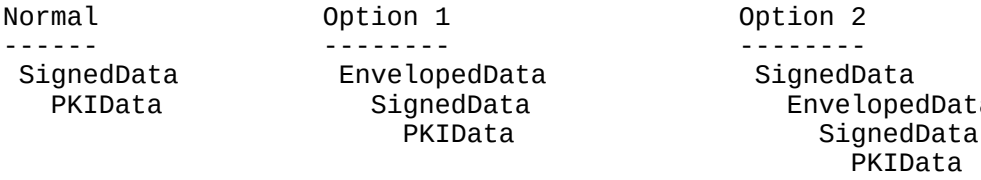
2. Additional processing of non-element items includes the saving of certificates and CRLs present in wrapping layers. This type of processing is based on the consumer of the element and should not be relied on by generators.

No processing is required for cmsSequence or otherMsgSequence members of the PKIResponse, if items are present and are not referenced by a control. In this case, the cmsSequence and otherMsgSequence members are to be ignored.

5. Application of Encryption to a PKI Request/Response

There are occasions when a PKI Request or Response must be encrypted in order to prevent disclosure of information in the PKI Request/ Response from being accessible to unauthorized entities. This section describes the means to encrypt Full PKI Requests and Responses (Simple PKI Requests cannot be encrypted). Data portions of PKI Requests and Responses that are placed in the cmsSequence field can be encrypted separately.

Confidentiality is provided by wrapping the PKI Request/Response (a SignedData) in an EnvelopedData. 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. It is recommended that if an EnvelopedData layer is applied to a PKI Request/Response, a second signature layer be placed outside of the EnvelopedData layer. The following figure shows how this nesting would be done:



Note: PKIResponse can be substituted for PKIData in the above figure.

Options 1 and 2 prevent leakage of sensitive data by encrypting the Full PKI Request/Response. An RA that receives a PKI Request that it cannot decrypt MAY reject the PKI Request unless it can process the PKI Request without knowledge of the contents (i.e., all it does is amalgamate multiple PKI Requests and forward them to a server).

After the RA removes the envelope and completes processing, it may then apply a new EnvelopedData layer to protect PKI Requests for transmission to the next processing agent. Section 7 contains more information about RA processing.

Full PKI Requests/Responses can be encrypted or transmitted in the clear. Servers MUST provide support for all three options.

Alternatively, an authenticated, secure channel could exist between the parties that require confidentiality. Clients and servers MAY use such channels instead of the technique described above to provide secure, private communication of Simple and Full PKI Requests/Responses.

6. Controls

Controls are carried as part of both Full PKI Requests and Responses. Each control is encoded as a unique OID followed by the data for the control (see syntax in Section 3.2.1.1). The encoding of the data is based on the control. Processing systems would first detect the OID (TaggedAttribute attrType) and process the corresponding control value (TaggedAttribute attrValues) prior to processing the message body.

The OIDs are all defined under the following arc:

```
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 }
```

The following table lists the names, OID, and syntactic structure for each of the controls described in this document.

Identifier Description	OID	ASN.1 Structure	Section
id-cmc-statusInfo	id-cmc 1	CMCStatusInfo	6.1.2
id-cmc-identification	id-cmc 2	UTF8String	6.2.3
id-cmc-identityProof	id-cmc 3	OCTET STRING	6.2.2
id-cmc-dataReturn	id-cmc 4	OCTET STRING	6.4
id-cmc-transactionId	id-cmc 5	INTEGER	6.6
id-cmc-senderNonce	id-cmc 6	OCTET STRING	6.6
id-cmc-recipientNonce	id-cmc 7	OCTET STRING	6.6
id-cmc-addExtensions	id-cmc 8	AddExtensions	6.5.2
id-cmc-encryptedPOP	id-cmc 9	EncryptedPOP	6.7
id-cmc-decryptedPOP	id-cmc 10	DecryptedPOP	6.7
id-cmc-lraPOPWitness	id-cmc 11	LraPOPWitness	6.8
id-cmc-getCert	id-cmc 15	GetCert	6.9

id-cmc-getCRL	id-cmc 16	GetCRL	6.10
id-cmc-revokeRequest	id-cmc 17	RevokeRequest	6.11
id-cmc-regInfo	id-cmc 18	OCTET STRING	6.12
id-cmc-responseInfo	id-cmc 19	OCTET STRING	6.12
id-cmc-queryPending	id-cmc 21	OCTET STRING	6.13
id-cmc-popLinkRandom	id-cmc 22	OCTET STRING	6.3.1
id-cmc-popLinkWitness	id-cmc 23	OCTET STRING	6.3.1
id-cmc-popLinkWitnessV2	id-cmc 33	OCTET STRING	6.3.1.1
id-cmc-confirmCertAcceptance	id-cmc 24	CMCCertId	6.14
id-cmc-statusInfoV2	id-cmc 25	CMCStatusInfoV2	6.1.1
id-cmc-trustedAnchors	id-cmc 26	PublishTrustAnchors	6.15
id-cmc-authData	id-cmc 27	AuthPublish	6.16
id-cmc-batchRequests	id-cmc 28	BodyPartList	6.17
id-cmc-batchResponses	id-cmc 29	BodyPartList	6.17
id-cmc-publishCert	id-cmc 30	CMCPublicationInfo	6.18
id-cmc-modCertTemplate	id-cmc 31	ModCertTemplate	6.5.1

id-cmc-controlProcessed	id-cmc 32	ControlsProcessed	6.19
id-cmc-identityProofV2	id-cmc 34	IdentityProofV2	6.2.1
id-cmc-raIdentityWitness	id-cmc 35	BodyPartPath	6.20
id-cmc-responseBody	id-cmc 37	BodyPartPath	6.21

Table 1: CMC Control Attributes

6.1. CMC Status Info Controls

The CMC Status Info controls return information about the status of a client/server request/response. Two controls are described in this section. The Extended CMC Status Info control is the preferred control; the CMC Status Info control 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 PKI Response. Servers MUST use the Extended CMC Status Info control, but MAY additionally use the CMC Status Info control. Clients MUST be able to process the Extended

CMC Status Info control.

6.1.1. Extended CMC Status Info Control

The Extended CMC Status Info control is identified by the OID:

```
id-cmc-statusInfoV2 OBJECT IDENTIFIER ::= { id-cmc 25 }
```

The Extended CMC Status Info control has the ASN.1 definition:

```
CMCStatusInfoV2 ::= SEQUENCE {
   cMCStatus             CMCStatus,
   bodyList              SEQUENCE SIZE (1..MAX) OF
                           BodyPartReference,
   statusString          UTF8String OPTIONAL,
   otherInfo             OtherStatusInfo OPTIONAL
}

OtherStatusInfo ::= CHOICE {
   failInfo              CMCFailInfo,
   pendInfo              PendInfo,
   extendedFailInfo      [1] ExtendedFailInfo
}

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

ExtendedFailInfo ::= SEQUENCE {
   failInfoOID            OBJECT IDENTIFIER,
   failInfoValue          ANY DEFINED BY failInfoOID
}

BodyPartReference ::= CHOICE {
  bodyPartID           BodyPartID,
  bodyPartPath         BodyPartPath
}
```

The fields in CMCStatusInfoV2 have the following meaning:

cMCStatus contains the returned status value. Details are in Section 6.1.3.

bodyList identifies the controls or other elements to which the status value applies. If an error is returned for a Simple PKI Request, this field is the bodyPartID choice of BodyPartReference with the single integer of value 1.

statusString contains additional description information. This string is human readable.

otherInfo contains additional information that expands on the CMC status code returned in the cMCStatus field.

The fields in OtherStatusInfo have the following meaning:

failInfo is described in Section 6.1.4. It provides an error code that details what failure occurred. This choice is present only if cMCStatus contains the value failed.

pendInfo contains information about when and how the client should request the result of this request. It is present when the cMCStatus is either pending or partial. pendInfo uses the structure PendInfo, which has the fields:

- pendToken is the token used in the Query Pending control (Section 6.13).
- pendTime contains the suggested time the server wants to be queried about the status of the certification request.

extendedFailInfo includes application-dependent detailed error information. This choice is present only if cMCStatus contains the value failed. Caution should be used when defining new values as they may not be correctly recognized by all clients and servers. The CMCFailInfo value of internalCAError may be assumed if the extended error is not recognized. This field uses the type ExtendedFailInfo. ExtendedFailInfo has the fields:

- failInfoOID contains an OID that is associated with a set of extended error values.
- failInfoValue contains an extended error code from the defined set of extended error codes.

If the cMCStatus field is success, the Extended CMC Status Info control MAY be omitted unless it is the only item in the response.

6.1.2. CMC Status Info Control

The CMC Status Info control is identified by the OID:

```
id-cmc-statusInfo OBJECT IDENTIFIER ::= { id-cmc 1 }
```

The CMC Status Info control has the ASN.1 definition:

```
CMCStatusInfo ::= SEQUENCE {
  cMCStatus           CMCStatus,
  bodyList            BodyPartList,
  statusString        UTF8String OPTIONAL,
  otherInfo           CHOICE {
    failInfo            CMCFailInfo,
    pendInfo            PendInfo } OPTIONAL
  }
```

The fields in CMCStatusInfo have the following meaning:

cMCStatus contains the returned status value. Details are in Section 6.1.3.

bodyList contains the list of controls or other elements to which the status value applies. If an error is being returned for a Simple PKI Request, this field contains a single integer of value 1.

statusString contains additional description information. This string is human readable.

otherInfo provides additional information that expands on the CMC status code returned in the cMCStatus field.

- failInfo is described in Section 6.1.4. It provides an error code that details what failure occurred. This choice is present only if cMCStatus is failed.

- pendInfo uses the PendInfo ASN.1 structure in Section 6.1.1. It contains information about when and how the client should request results of this request. The pendInfo field MUST be populated for a cMCStatus value of pending or partial. Further details can be found in Section 6.1.1 (Extended CMC Status Info Control) and Section 6.13 (Query Pending Control).

If the cMCStatus field is success, the CMC Status Info control MAY be omitted unless it is the only item in the response. If no status exists for a Simple or Full PKI Request, then the value of success is assumed.

6.1.3. CMCStatus Values

CMCStatus is a field in the Extended CMC Status Info and CMC Status Info controls. This field contains a code representing the success or failure of a specific operation. CMCStatus has the ASN.1 structure:

```
CMCStatus ::= INTEGER {
  success                (0),
  -- reserved            (1),
  failed                 (2),
  pending                (3),
  noSupport              (4),
  confirmRequired        (5),
  popRequired            (6),
  partial                (7)
  }
```

The values of CMCStatus have the following meaning:

success indicates the request was granted or the action was completed.

failed indicates the request was not granted or the action was not completed. More information is included elsewhere in the response.

pending indicates the PKI Request has yet to be processed. The requester is responsible to poll back on this Full PKI request. pending may only be returned for certification request operations.

noSupport indicates the requested operation is not supported.

confirmRequired indicates a Confirm Certificate Acceptance control (Section 6.14) must be returned before the certificate can be used.

popRequired indicates a direct POP operation is required (Section 6.3.1.3).

partial indicates a partial PKI Response is returned. The requester is responsible to poll back for the unfulfilled portions of the Full PKI Request.

6.1.4. CMCFailInfo

CMCFailInfo is a field in the Extended CMC Status Info and CMC Status Info controls. CMCFailInfo conveys more detailed information relevant to the interpretation of a failure condition. The CMCFailInfo has the following ASN.1 structure:

```
CMCFailInfo ::= INTEGER {
  badAlg            (0),
  badMessageCheck   (1),
  badRequest        (2),
  badTime           (3),
  badCertId         (4),
  unsupportedExt     (5),
  mustArchiveKeys   (6),
  badIdentity       (7),
  popRequired       (8),
  popFailed         (9),
  noKeyReuse        (10),
  internalCAError   (11),
  tryLater          (12),
  authDataFail      (13)
  }
```

The values of CMCFailInfo have the following meanings:

badAlg indicates unrecognized or unsupported algorithm.

badMessageCheck indicates integrity check failed.

badRequest indicates transaction was not permitted or supported.

badTime indicates message time field was not sufficiently close to the system time.

badCertId indicates no certificate could be identified matching the provided criteria.

unsupportedExt indicates a requested X.509 extension is not supported by the recipient CA.

mustArchiveKeys indicates private key material must be supplied.

badIdentity indicates identification control failed to verify.

popRequired indicates server requires a POP proof before issuing certificate.

popFailed indicates POP processing failed.

noKeyReuse indicates server policy does not allow key reuse.

internalCAError indicates that the CA had an unknown internal failure.

tryLater indicates that the server is not accepting requests at this time and the client should try at a later time.

authDataFail indicates failure occurred during processing of authenticated data.

If additional failure reasons are needed, they SHOULD use the ExtendedFailureInfo item in the Extended CMC Status Info control. However, for closed environments they can be defined using this type. Such codes MUST be in the range from 1000 to 1999.

6.2. Identification and Identity Proof Controls

Some CAs and RAs 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 are familiar with a bank's request to provide your mother's maiden name as a form of identity proof. The reasoning behind requiring a proof-of-identity can be found in Appendix C of [CRMF].

CMC provides a method to prove the client's identity based on a client/server shared-secret. If clients support the Full PKI Request, clients MUST implement this method of identity proof (Section 6.2.2). Servers MUST provide this method, but MAY additionally support bilateral methods of similar strength.

This document also provides an Identification control (Section 6.2.3). This control is a simple method to allow a client to state who they are to the server. Generally, a shared-secret AND an identifier of that shared-secret are passed from the server to the client. The identifier is placed in the Identification control, and the shared-secret is used to compute the Identity Proof control.

6.2.1. Identity Proof Version 2 Control

The Identity Proof Version 2 control is identified by the OID:

```
id-cmc-identityProofV2 OBJECT IDENTIFIER ::= { id-cmc 34 }
```

The Identity Proof Version 2 control has the ASN.1 definition:

```
IdentifyProofV2 ::= SEQUENCE {
  hashAlgID        AlgorithmIdentifier,
  macAlgID         AlgorithmIdentifier,
  witness          OCTET STRING
  }
```

The fields of IdentityProofV2 have the following meaning:

hashAlgID is the identifier and parameters for the hash algorithm used to convert the shared-secret into a key for the MAC algorithm.

macAlgID is the identifier and the parameters for the message authentication code algorithm used to compute the value of the witness field.

witness is the identity proof.

The required 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 PKIData reqSequence field (encoded exactly as it appears in the Full PKI Request including the sequence type and length) is the value to be validated.

2. A hash of the shared-secret as a UTF8 string is computed using hashAlgID.

3. A MAC is then computed using the value produced in Step 1 as the message and the value from Step 2 as the key.

4. The result from Step 3 is then encoded as the witness value in the Identity Proof Version 2 control.

When the server verifies the Identity Proof Version 2 control, it computes the MAC value in the same way and compares it to the witness value contained in the PKI Request.

If a server fails the verification of an Identity Proof Version 2 control, the CMCFailInfo value MUST be present in the Full PKI Response and MUST have a value of badIdentity.

Reuse of the shared-secret on certification request retries allows the client and server to maintain the same view of acceptable identity proof values. However, reuse of the shared-secret can potentially open the door for some types of attacks.

Implementations MUST be able to support tokens at least 16 characters long. Guidance on the amount of entropy actually obtained from a given length token based on character sets can be found in Appendix A of [PASSWORD].

6.2.2. Identity Proof Control

The Identity Proof control is identified by the OID:

```
id-cmc-identityProof OBJECT IDENTIFIER ::= { id-cmc 3 }
```

The Identity Proof control has the ASN.1 definition:

```
IdentifyProof ::= OCTET STRING
```

This control is processed in the same way as the Identity Proof Version 2 control. In this case, the hash algorithm is fixed to SHA-1 and the MAC algorithm is fixed to HMAC-SHA1.

6.2.3. Identification Control

Optionally, servers MAY require the inclusion of the unprotected Identification control with an Identification Proof control. The Identification control is intended to contain a text string that assists the server in locating the shared-secret needed to validate the contents of the Identity Proof control. If the Identification control is included in the Full PKI Request, the derivation of the key in Step 2 (from Section 6.2.1) is altered so that the hash of the concatenation of the shared-secret and the UTF8 identity value (without the type and length bytes) are hashed rather than just the shared-secret.

The Identification control is identified by the OID:

```
id-cmc-identification OBJECT IDENTIFIER ::= { id-cmc 2 }
```

The Identification control has the ASN.1 definition:

```
Identification ::= UTF8String
```

6.2.4. Hardware Shared-Secret Token Generation

The shared-secret between the EE and the server is sometimes computed using a hardware device that generates a series of tokens. The EE can therefore prove its 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 control 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.

4. The entire certification request MUST be shrouded in some fashion to prevent eavesdropping. Although the token is time critical, an active eavesdropper cannot be permitted to extract the token and submit a different certification request with the same token value.

6.3. Linking Identity and POP Information

In a CMC Full PKI Request, identity proof information about the client is carried in the certificate associated with the signature of the SignedData containing the certification requests, one of the two identity proof controls or the MAC computed for the AuthenticatedData containing the certification requests. Proof-of-possession (POP) information for key pairs, however, is carried separately for each PKCS #10 or CRMF certification request. (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 6.7 are used.) In order to prevent substitution-style attacks, the protocol must guarantee that the same entity supplied both the POP and proof-of-identity information.

We describe three mechanisms for linking identity and POP information: witness values cryptographically derived from a shared-secret (Section 6.3.1), shared-secret/subject name matching (Section 6.3.2) , and subject name matching to an existing certificate (Section 6.3.3) . Clients and servers MUST support the witness value and the certificate linking techniques. Clients and servers MAY support shared-secret/name matching or MAY support other bilateral techniques of similar strength. The idea behind the first

two mechanisms is to force the client to sign some data into each certification request that can be directly associated with the shared-secret; this will defeat attempts to include certification requests from different entities in a single Full PKI Request.

6.3.1. Cryptographic Linkage

The first technique that links identity and POP information forces the client to include a piece of information cryptographically derived from the shared-secret as a signed extension within each certification request (PKCS #10 or CRMF).

6.3.1.1. POP Link Witness Version 2 Controls

The POP Link Witness Version 2 control is identified by the OID:

```
id-cmc-popLinkWitnessV2 OBJECT IDENTIFIER ::= { id-cmc 33 }
```

The POP Link Witness Version 2 control has the ASN.1 definition:

```
PopLinkWitnessV2 ::= SEQUENCE {
  keyGenAlgorithm   AlgorithmIdentifier,
  macAlgorithm      AlgorithmIdentifier,
  witness           OCTET STRING
  }
```

The fields of PopLinkWitnessV2 have the following meanings:

keyGenAlgorithm contains the algorithm used to generate the key for the MAC algorithm. This will generally be a hash algorithm, but could be a more complex algorithm.

macAlgorithm contains the algorithm used to create the witness value.

witness contains the computed witness value.

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 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. The key is computed from the shared-secret using the algorithm in keyGenAlgorithm.

3. A MAC is then computed over the random value produced in Step 1, using the key computed in Step 2.

4. The random value produced in Step 1 is encoded as the value of a POP Link Random control. This control MUST be included in the Full PKI Request.

5. The MAC value produced in Step 3 is placed in either the POP Link Witness control or the witness field of the POP Link Witness V2 control.

   * For CRMF, the POP Link Witness/POP Link Witness V2 control is included in the controls field of the CertRequest structure.

   * For PKCS #10, the POP Link Witness/POP Link Witness V2 control is included in the attributes field of the CertificationRequestInfo structure.

Upon receipt, servers MUST verify that each certification request contains a copy of the POP Link Witness/POP Link Witness V2 control and that its value was derived using the above method from the shared-secret and the random string included in the POP Link Random control.

The Identification control (Section 6.2.3) or the subject DN of a certification request can be used to help identify which shared-secret was used.

6.3.1.2. POP Link Witness Control

The POP Link Witness control is identified by the OID:

```
id-cmc-popLinkWitness OBJECT IDENTIFIER ::= { id-cmc 23 }
```

The POP Link Witness control has the ASN.1 definition:

```
PopLinkWitness ::= OCTET STRING
```

For this control, SHA-1 is used as the key generation algorithm. HMAC-SHA1 is used as the mac algorithm.

6.3.1.3. POP Link Random Control

The POP Link Random control is identified by the OID:

```
id-cmc-popLinkRandom OBJECT IDENTIFIER ::= { id-cmc 22 }
```

The POP Link Random control has the ASN.1 definition:

```
PopLinkRandom ::= OCTET STRING
```

6.3.2. Shared-Secret/Subject DN Linking

The second technique to link identity and POP information 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 certification request. It is expected that many client-server connections that use shared-secret-based proof-of-identity will use this mechanism. (It is common not to omit the subject DN information from the certification request.)

When the shared-secret is generated and transferred out-of-band to initiate the registration process (Section 6.2), 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 the 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, 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 certification request (PKCS #10 and/or CRMF) in the Full PKI Request. The subject names in the certification requests MUST NOT be null.

2. The client MUST include an Identity Proof control (Section 6.2.2) or Identity Proof Version 2 control (Section 6.2.1), derived from the shared-secret, in the Full PKI Request.

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

6.3.3. Existing Certificate Linking

Linking between the POP and an identity is easy when an existing certificate is used. The client copies all of the naming information from the existing certificate (subject name and subject alternative name) into the new certification request. The POP on the new public key is then performed by using the new key to sign the identity information (linking the POP to a specific identity). The identity information is then tied to the POP information by signing the entire enrollment request with the private key of the existing certificate.

Existing certificate linking can be used in the following circumstances:

When replacing a certificate by doing a renewal or rekey certification request.

Using an existing certificate to get a new certificate. An example of this would be to get a key establishment certificate after having gotten a signature certificate.

Using a third-party certificate to get a new certificate from a CA. An example of this would be using a certificate and key pair distributed with a device to prove an identity. This requires that the CA have an out-of-band channel to map the identity in the device certificate to the new EE identity.

6.4. Data Return Control

The Data Return control 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 Full PKI Response. Data placed in a Data Return control is considered to be opaque to the server. The same control is used for both Full PKI Requests and Responses. If the Data Return control appears in a Full PKI Request, the server MUST return it as part of the PKI Response.

In the event that the information in the Data Return control 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.

The Data Return control is identified by the OID:

```
id-cmc-dataReturn OBJECT IDENTIFIER ::= { id-cmc 4 }
```

The Data Return control has the ASN.1 definition:

```
DataReturn ::= OCTET STRING
```

A client could use this control 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.

6.5. RA Certificate Modification Controls

These controls exist for RAs to be able to modify the contents of a certification request. Modifications might be necessary for various reasons. These include addition of certificate extensions or modification of subject and/or subject alternative names.

Two controls exist for this purpose. The first control, Modify Certification Request (Section 6.5.1), allows the RA to replace or remove any field in the certificate. The second control, Add Extensions (Section 6.5.2), only allows for the addition of extensions.

6.5.1. Modify Certification Request Control

The Modify Certification Request control is used by RAs to change fields in a requested certificate.

The Modify Certification Request control is identified by the OID:

```
id-cmc-modCertTemplate OBJECT IDENTIFIER ::= { id-cmc 31 }
```

The Modify Certification Request has the ASN.1 definition:

```
ModCertTemplate ::= SEQUENCE {
  pkiDataReference            BodyPartPath,
  certReferences              BodyPartList,
  replace                     BOOLEAN DEFAULT TRUE,
  certTemplate                CertTemplate
  }
```

The fields in ModCertTemplate have the following meaning:

pkiDataReference is the path to the PKI Request containing certification request(s) to be modified.

certReferences refers to one or more certification requests in the PKI Request referenced by pkiDataReference to be modified. Each BodyPartID of the certReferences sequence MUST be equal to either the bodyPartID of a TaggedCertificationRequest (PKCS #10) or the certReqId of the CertRequest within a CertReqMsg (CRMF). By definition, the certificate extensions included in the certTemplate field are applied to every certification request referenced in the certReferences sequence. If a request corresponding to bodyPartID cannot be found, the CMCFailInfo with a value of badRequest is returned that references this control.

replace specifies if the target certification request is to be modified by replacing or deleting fields. If the value is TRUE, the data in this control replaces the data in the target certification request. If the value is FALSE, the data in the target certification request is deleted. The action is slightly different for the extensions field of certTemplate; each extension is treated individually rather than as a single unit.

certTemplate is a certificate template object [CRMF]. If a field is present and replace is TRUE, it replaces that field in the certification request. If the field is present and replace is FALSE, the field in the certification request is removed. If the field is absent, no action is performed. Each extension is treated as a single field.

Servers MUST be able to process all extensions defined, but not prohibited, in [PKIXCERT]. Servers are not required to be able to process every X.509v3 extension transmitted using this protocol, nor are they required to be able to process other, private extensions. Servers are not required to put all RA-requested extensions into a certificate. Servers are permitted to modify RA-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 Full PKI Response is returned, the server MUST return a CMCFailInfo value with the value of unsupportedExt.

If a certification request is the target of multiple Modify Certification Request controls, the behavior is:

* If control A exists in a layer that contains the layer of control B, control A MUST override control B. In other words, controls should be applied from the innermost layer to the outermost layer.

* If control A and control B are in the same PKIData (i.e., the same wrapping layer), the order of application is non-determinate.

The same order of application is used if a certification request is the target of both a Modify Certification Request control and an Add Extensions control.

6.5.2. Add Extensions Control

The Add Extensions control has been deprecated in favor of the Modify Certification Request control. It was replaced so that fields in the certification request other than extensions could be modified.

The Add Extensions control is used by RAs to specify additional extensions that are to be included in certificates.

The Add Extensions control is identified by the OID:

```
id-cmc-addExtensions OBJECT IDENTIFIER ::= { id-cmc 8 }
```

The Add Extensions control has the ASN.1 definition:

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

The fields in AddExtensions have the following meaning:

pkiDataReference contains the body part identity of the embedded certification request.

certReferences is a list of references to one or more of the certification requests contained within a PKIData. Each body part identifier of the certReferences sequence MUST be equal to either the bodyPartID of a TaggedCertificationRequest (PKCS #10) or the certReqId of the CertRequest within a CertReqMsg (CRMF). By definition, the listed extensions are to be applied to every certification request referenced in the certReferences sequence. If a certification request corresponding to bodyPartID cannot be found, the CMCFailInfo with a value of badRequest is returned referencing this control.

extensions is a sequence of extensions to be applied to the referenced certification requests.

Servers MUST be able to process all extensions defined, but not prohibited, in [PKIXCERT]. Servers are not required to be able to process every X.509v3 extension transmitted using this protocol, nor are they required to be able to process other, private extensions. Servers are not required to put all RA-requested extensions into a certificate. Servers are permitted to modify RA-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 CMCFailInfo with the value of unsupportedExt.

If multiple Add Extensions controls exist in a Full PKI Request, the exact behavior is left up to the CA policy. However, it is recommended that the following policy be used. These rules would be applied to individual extensions within an Add Extensions control (as opposed to an "all or nothing" approach).

1. If the conflict is within a single PKIData, the certification request would be rejected with a CMCFailInfo value of badRequest.

2. If the conflict is between different PKIData, the outermost version of the extension would be used (allowing an RA to override the requested extension).

6.6. Transaction Identifier Control and Sender and Recipient Nonce Controls

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

The Transaction Identifier control is identified by the OID:

```
id-cmc-transactionId  OBJECT IDENTIFIER ::= { id-cmc 5 }
```

The Transaction Identifier control has the ASN.1 definition:

```
TransactionId ::= INTEGER
```

The Transaction Identifier control identifies a given transaction. It is used by client and server to manage the state of an operation. Clients MAY include a Transaction Identifier control in a request. If the original request contains a Transaction Identifier control, all subsequent requests and responses MUST include the same Transaction Identifier control.

Replay protection is supported through the use of the Sender and Recipient Nonce controls. If nonces are used, in the first message of a transaction, a Recipient Nonce control is not transmitted; a Sender Nonce control is included by the transaction originator and retained for later reference. The recipient of a Sender Nonce control reflects this value back to the originator as a Recipient Nonce control and includes its own Sender Nonce control. Upon receipt by the transaction originator of this response, the transaction originator compares the value of Recipient Nonce control to its retained value. If the values match, the message can be accepted for further security processing. The received value for a Sender Nonce control is also retained for inclusion in the next message associated with the same transaction.

The Sender Nonce and Recipient Nonce controls are identified by the OIDs:

```
id-cmc-senderNonce    OBJECT IDENTIFIER ::= { id-cmc 6 }
id-cmc-recipientNonce OBJECT IDENTIFIER ::= { id-cmc 7 }
```

The Sender Nonce control has the ASN.1 definition:

```
SenderNonce ::= OCTET STRING
```

The Recipient Nonce control has the ASN.1 definition:

```
RecipientNonce ::= OCTET STRING
```

Clients MAY include a Sender Nonce control in the initial PKI Request. If a message includes a Sender Nonce control, the response MUST include the transmitted value of the previously received Sender Nonce control as a Recipient Nonce control and include a new value as its Sender Nonce control.

6.7. Encrypted and Decrypted POP Controls

Servers MAY require that this POP method be used only if another POP method is unavailable. Servers SHOULD reject all certification requests contained within a PKIData if any required POP is missing for any element within the PKIData.

Many servers require proof that the entity that generated the certification request 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 steps:

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.

3. Client decrypts the POP challenge using the private key that corresponds to the presented public key and uses it for computing a keyed hash value sent back to the server.

4. Server validates the decrypted POP challenge and continues processing the certification request.

CMC defines two different controls. The first deals with the encrypted challenge sent from the server to the user in Step 2. The second deals with the value derived from the decrypted challenge sent by the client to the server in Step 3.

The Encrypted POP control is used to send the encrypted challenge from the server to the client as part of the PKIResponse. (Note that it is assumed that the message sent in Step 1 above is a Full PKI Request and that the response in Step 2 is a Full PKI Response including a CMCFailInfo specifying that a POP is explicitly required, and providing the POP challenge in the encryptedPOP control.)

The Encrypted POP control is identified by the OID:

```
id-cmc-encryptedPOP OBJECT IDENTIFIER ::= { id-cmc 9 }
```

The Encrypted POP control has the ASN.1 definition:

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

The Decrypted POP control is identified by the OID:

```
  id-cmc-decryptedPOP  OBJECT IDENTIFIER ::= { id-cmc 10 }
```

The Decrypted POP control has the ASN.1 definition:

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

The encrypted POP algorithm works as follows:

1. The server randomly generates the POP Proof Value and associates it with the request.

2. The server returns the Encrypted POP control with the following fields set:

   * request is the original certification request (it is included here so the client need not keep a copy of the request).

   * cms is an EnvelopedData, the encapsulated content type being id- data and the content being the POP Proof Value; this value needs to be long enough that one cannot reverse the value from the witness hash. If the certification request contains a Subject Key Identifier (SKI) extension, then the recipient identifier SHOULD be the SKI. If the issuerAndSerialNumber form is used, the IssuerName MUST be encoded as NULL and the SerialNumber as the bodyPartID of the certification request.

   * thePOPAlgID identifies the algorithm to be used in computing the return POP value.

   * witnessAlgID identifies the hash algorithm used on the POP Proof Value to create the field witness.

   * witness is the hashed value of the POP Proof Value.

3. The client decrypts the cms field to obtain the POP Proof Value. The client computes H(POP Proof Value) 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 containing a CMC Status Info control with CMCFailInfo value of popFailed.

4. The client creates the Decrypted POP control as part of a new PKIData. The fields in the DecryptedPOP are:

   * bodyPartID refers to the certification request in the new PKI Request.

   * thePOPAlgID is copied from the encryptedPOP.

   * thePOP contains the possession proof. This value is computed by thePOPAlgID using the POP Proof Value and the request.

5. The server then re-computes the value of thePOP from its cached value 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. The POP Proof Value is used as the secret value in the HMAC algorithm and the request is used as the data. If the POP Proof Value is greater than 64 bytes, only the first 64 bytes of the POP Proof Value is 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. The following 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 certification request R, server computes y = F(x,R). F can be, for example, HMAC-SHA256(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 certification request structure. y should not be predictable based on knowledge of R, thus the use of a one-way function like HMAC-SHA256.

6.8. RA POP Witness Control

In a certification request scenario that involves an RA, the CA may allow (or require) that the RA perform the POP protocol with the entity that generated the certification request. In this case, the RA needs a way to inform the CA that it has done the POP. The RA POP Witness control addresses this issue.

The RA POP Witness control is identified by the OID:

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

The RA POP Witness control has the ASN.1 definition:

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

The fields in LraPOPWitness have the following meaning:

pkiDataBodyid contains the body part identifier of the nested TaggedContentInfo containing the client's Full PKI Request. pkiDataBodyid is set to 0 if the request is in the current PKIData.

bodyIds is a list of certification requests for which the RA 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 certification server does not allow an RA to do the POP verification, it returns a CMCFailInfo with the value of popFailed. The CA MUST NOT start a challenge-response to re-verify the POP itself.

6.9. Get Certificate Control

Everything described in this section is optional to implement.

The Get Certificate control is used to retrieve a previously issued certificate from a certificate repository. A CA, an RA, 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 Certificate control is identified by the OID:

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

The Get Certificate control has the ASN.1 definition:

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

The fields in GetCert have the following meaning:

issuerName is the name of the certificate issuer.

serialNumber identifies the certificate to be retrieved.

The server that responds to this request places the requested certificate in the certificates field of a SignedData. If the Get Certificate control is the only control in a Full PKI Request, the response should be a Simple PKI Response.

6.10. Get CRL Control

Everything described in this section is optional to implement.

The Get CRL control is used to retrieve CRLs from a repository of CRLs. A CA, an RA, 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 is identified by the OID:

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

The Get CRL control has the ASN.1 definition:

```
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 server responding to this request places the requested CRL in the crls field of a SignedData. If the Get CRL control is the only control in a Full PKI Request, the response should be a Simple PKI Response.

6.11. Revocation Request Control

The Revocation Request control is used to request that a certificate be revoked.

The Revocation Request control is identified by the OID:

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

The Revocation Request control has the ASN.1 definition:

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

The fields of RevokeRequest have the following meaning:

issuerName is the issuerName of the certificate to be revoked.

serialNumber is the serial number of the certificate to be revoked.

reason is 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 is the suggested value for the Invalidity Date CRL Extension. The CA can use this value at its discretion in building the CRL.

sharedSecret is 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 is a human-readable comment.

For a revocation request to be reliable in the event of a dispute, a strong proof-of-origin is required. However, in the instance when an EE has lost use of its signature private key, it is impossible for the EE to produce a digital signature (prior to the certification of a new signature key pair). The Revoke Request control allows the EE to send the CA a shared-secret that may be used as an alternative authenticator in the instance of loss of use of the EE's signature private key. The acceptability of this practice is a matter of local security policy.

It is possible to sign the revocation for the lost certificate with a different certificate in some circumstances. A client can sign a revocation for an encryption key with a signing certificate if the name information matches. Similarly, an administrator or RA can be assigned the ability to revoke the certificate of a third party. Acceptance of the revocation by the server depends on local policy in these cases.

Clients MUST provide the capability to produce a digitally signed Revocation Request control. Clients SHOULD be capable of producing an unsigned Revocation Request control containing the EE shared-secret (the unsigned message consisting of a SignedData with no signatures). If a client provides shared-secret-based self-revocation, the client MUST be capable of producing a Revocation Request control 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 control. The shared-secret has a one-time use (i.e., it is used to request

revocation of the certificate), 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 PKI Response MUST be returned for a revocation request.

6.12. Registration and Response Information Controls

The Registration Information control allows for clients to pass additional information as part of a Full PKI Request.

The Registration Information control is identified by the OID:

```
id-cmc-regInfo OBJECT IDENTIFIER ::= { id-cmc 18 }
```

The Registration Information control has the ASN.1 definition:

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

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

The Response Information control allows a server to return additional information as part of a Full PKI Response.

The Response Information control is identified by the OID:

```
id-cmc-responseInfo  OBJECT IDENTIFIER ::= { id-cmc 19 }
```

The Response Information control has the ASN.1 definition:

```
ResponseInfo ::= OCTET STRING
```

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

6.13. Query Pending Control

In some environments, process requirements for manual intervention or other identity checks can delay the return of the certificate. The Query Pending control allows clients to query a server about the state of a pending certification request. The server returns a pendToken as part of the Extended CMC Status Info and the CMC Status Info controls (in the otherInfo field). The client copies the pendToken into the Query Pending control to identify the correct certification request to the server. The server returns a suggested time for the client to query for the state of a pending certification

request.

The Query Pending control is identified by the OID:

```
id-cmc-queryPending  OBJECT IDENTIFIER ::= { id-cmc 21 }
```

The Query Pending control has the ASN.1 definition:

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

If a server returns a pending or partial CMCStatusInfo (the transaction is still pending), the otherInfo MAY be omitted. If the otherInfo is not omitted, the value of 'pendInfo' MUST be the same as the original pendInfo value.

6.14. Confirm Certificate Acceptance Control

Some CAs require that clients give a positive confirmation that the certificates issued to the EE are acceptable. The Confirm Certificate Acceptance control is used for that purpose. If the CMC Status Info on a PKI Response is confirmRequired, then the client MUST return a Confirm Certificate Acceptance control contained in a Full PKI Request.

Clients SHOULD wait for the PKI Response from the server that the confirmation has been received before using the certificate for any purpose.

The Confirm Certificate Acceptance control is identified by the OID:

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

The Confirm Certificate Acceptance control has the ASN.1 definition:

```
CMCCertId ::= IssuerAndSerialNumber
```

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

Servers MUST return a Full PKI Response for a Confirm Certificate Acceptance control.

Note that if the CA includes this control, there will be two full round trips of messages.

1. The client sends the certification request to the CA.

2. The CA returns a Full PKI Response with the certificate and this control.

3. The client sends a Full PKI Request to the CA with an Extended CMC Status Info control accepting and a Confirm Certificate Acceptance control or an Extended CMC Status Info control rejecting the certificate.

4. The CA sends a Full PKI Response to the client with an Extended CMC Status Info of success.

6.15. Publish Trust Anchors Control

The Publish Trust Anchors control allows for the distribution of set trust anchors from a central authority to an EE. The same control is also used to update the set of trust anchors. Trust anchors are distributed in the form of certificates. These are expected, but not required, to be self-signed certificates. Information is extracted from these certificates to set the inputs to the certificates validation algorithm in Section 6.1.1 of [PKIXCERT].

The Publish Trust Anchors control is identified by the OID:

```
id-cmc-trustedAnchors  OBJECT IDENTIFIER ::= { id-cmc 26 }
```

The Publish Trust Anchors control has the ASN.1 definition:

```
PublishTrustAnchors ::= SEQUENCE {
  seqNumber      INTEGER,
  hashAlgorithm  AlgorithmIdentifier,
  anchorHashes   SEQUENCE OF OCTET STRING
}
```

The fields in PublishTrustAnchors have the following meaning:

seqNumber is an integer indicating the location within a sequence of updates.

hashAlgorithm is the identifier and parameters for the hash algorithm that is used in computing the values of the anchorHashes field. All implementations MUST implement SHA-256 for this field.

anchorHashes are the hashes for the certificates that are to be treated as trust anchors by the client. The actual certificates are transported in the certificate bag of the containing SignedData structure.

While it is recommended that the sender place the certificates that are to be trusted in the PKI Response, it is not required as the certificates should be obtainable using normal discovery techniques.

Prior to accepting the trust anchors changes, a client MUST at least do the following: validate the signature on the PKI Response to a current trusted anchor, check with policy to ensure that the signer is permitted to use the control, validate that the authenticated publish time in the signature is near to the current time, and validate that the sequence number is greater than the previously used one.

In the event that multiple agents publish a set of trust anchors, it is up to local policy to determine how the different trust anchors should be combined. Clients SHOULD be able to handle the update of multiple trust anchors independently.

Note: Clients that handle this control must use extreme care in validating that the operation is permissible. Incorrect handling of this control allows for an attacker to change the set of trust anchors on the client.

6.16. Authenticated Data Control

The Authenticated Data control allows a server to provide data back to the client in an authenticated manner. This control uses the Authenticated Data structure to allow for validation of the data. This control is used where the client has a shared-secret and a secret identifier with the server, but where a trust anchor has not yet been downloaded onto the client so that a signing certificate for the server cannot be validated. The specific case that this control was created for use with is the Publish Trust Anchors control (Section 6.15), but it may be used in other cases as well.

The Authenticated Data control is identified by the OID:

```
id-cmc-authData OBJECT IDENTIFIER ::= { id-cmc 27 }
```

The Authenticated Data control has the ASN.1 definition:

```
AuthPublish ::= BodyPartID
```

AuthPublish is a body part identifier that refers to a member of the cmsSequence element for the current PKI Response or PKI Data. The cmsSequence element is AuthenticatedData. The encapsulated content is an id-cct-PKIData. The controls in the controlSequence need to be processed if the authentication succeeds. (One example is the Publish Trust Anchors control in Section 6.15.)

If the authentication operation fails, the CMCFailInfo authDataFail is returned.

6.17. Batch Request and Response Controls

These controls allow for an RA to collect multiple requests together into a single Full PKI Request and forward it to a CA. The server would then process the requests and return the results in a Full PKI Response.

The Batch Request control is identified by the OID:

```
id-cmc-batchRequests OBJECT IDENTIFIER ::= { id-cmc 28 }
```

The Batch Response control is identified by the OID:

```
id-cmc-batchResponses OBJECT IDENTIFIER ::= { id-cmc 29 }
```

Both the Batch Request and Batch Response controls have the ASN.1 definition:

```
BodyPartList ::= SEQUENCE of BodyPartID
```

The data associated with these controls is a set of body part identifiers. Each request/response is placed as an individual entry in the cmcSequence of the new PKIData/PKIResponse. The body part identifiers of these entries are then placed in the body part list associated with the control.

When a server processes a Batch Request control, it MAY return the responses in one or more PKI Responses. A CMCStatus value of partial is returned on all but the last PKI Response. The CMCStatus would be success if the Batch Requests control was processed; the responses are created with their own CMCStatus code. Errors on individual requests are not propagated up to the top level.

When a PKI Response with a CMCStatus value of partial is returned, the Query Pending control (Section 6.13) is used to retrieve additional results. The returned status includes a suggested time after which the client should ask for the additional results.

6.18. Publication Information Control

The Publication Information control allows for modifying publication of already issued certificates, both for publishing and removal from publication. A common usage for this control is to remove an existing certificate from publication during a rekey operation. This control should always be processed after the issuance of new certificates and revocation requests. This control should not be processed if a certificate failed to be issued.

The Publication Information control is identified by the OID:

```
id-cmc-publishCert OBJECT IDENTIFIER ::= { id-cmc 30 }
```

The Publication Information control has the ASN.1 definition:

```
CMCPublicationInfo ::= SEQUENCE {
  hashAlg     AlgorithmIdentifier,
  certHashes      SEQUENCE of OCTET STRING,
  pubInfo         PKIPublicationInfo
  }

PKIPublicationInfo ::= SEQUENCE {
  action     INTEGER {
                    dontPublish (0),
                    pleasePublish (1) },
  pubInfos  SEQUENCE SIZE (1..MAX) OF SinglePubInfo OPTIONAL }

  -- pubInfos MUST NOT be present if action is "dontPublish"
  -- (if action is "pleasePublish" and pubInfos is omitted,
  -- "dontCare" is assumed)

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

The fields in CMCPublicationInfo have the following meaning:

hashAlg is the algorithm identifier of the hash algorithm used to compute the values in certHashes.

certHashes are the hashes of the certificates for which publication is to change.

pubInfo is the information where and how the certificates should be published. The fields in pubInfo (taken from [CRMF]) have the following meanings:

- o action indicates the action the service should take. It has two values:
  - + dontPublish indicates that the PKI should not publish the certificate (this may indicate that the requester intends to publish the certificate him/herself). dontPublish has the added connotation of removing from publication the certificate if it is already published.
  - + pleasePublish indicates that the PKI MAY publish the certificate using whatever means it chooses unless pubInfos is present. Omission of the CMC Publication Info control results in the same behavior.
- o pubInfos pubInfos indicates how (e.g., X500, Web, IP Address) the PKI SHOULD publish the certificate.

A single certificate SHOULD NOT appear in more than one Publication Information control. The behavior is undefined in the event that it does.

6.19. Control Processed Control

The Control Processed control allows an RA to indicate to subsequent control processors that a specific control has already been processed. This permits an RA in the middle of a processing stream to process a control defined either in a local context or in a subsequent document.

The Control Processed control is identified by the OID:

```
id-cmc-controlProcessed  OBJECT IDENTIFIER ::= { id-cmc 32 }
```

The Control Processed control has the ASN.1 definition:

```
ControlList ::= SEQUENCE {
  bodyList        SEQUENCE SIZE (1..MAX) OF BodyPartReference
}
```

bodyList is a series of body part identifiers that form a path to each of the controls that were processed by the RA. This control is only needed for those controls that are not part of this standard and thus would cause an error condition of a server attempting to deal with a control not defined in this

document. No error status is needed since an error causes the RA to return the request to the client with the error rather than passing the request on to the next server in the processing list.

6.20. RA Identity Proof Witness Control

The RA Identity Proof Witness control allows an RA to indicate to subsequent control processors that all of the identity proof requirements have been met. This permits the identity proof to be performed at a location closer to the end-entity. For example, the identity proof could be done at multiple physical locations, while the CA could operate on a company-wide basis. The RA performs the identity proof, and potentially other tasks that require the secret to be used, while the CA is prevented from knowing the secret. If the identity proof fails, then the RA returns an error to the client denoting that fact.

The RA Identity Proof Witness control is identified by the OID:

```
id-cmc-raIdentityWitness OBJECT IDENTIFIER ::= { id-cmc 35 }
```

The RA Identity Proof Witness control has the ASN.1 definition:

```
cmc-raIdentityWitness CMC-CONTROL ::=
  { BodyPartPath IDENTIFIED BY id-cmc-raIdentityWitness }
```

cmc-raIdentityWitness is a CMC-CONTROL associating the object identifier id-cmc-raIdentityWitness and the type BodyPartPath. This object is omitted from the 1988 module. The object is added to the object set Cmc-Control-Set. The control is permitted to appear only in the control sequence of a PKIData object. It MUST NOT appear in the control sequence of a PKIResponse. The control is permitted to be used only by an RA. The control may appear multiple times in a control sequence with each occurrence pointing to a different object.

id-cmc-raIdentityWitness is the object identifier used to identify this CMC control.

BodyPartPath is the type structure associated with the control. The syntax of BodyPartPath is defined in Section 3.2.2. The path contains a sequence of body part identifiers leading to one of the following items:

- Identity Proof control if the RA verified the identity proof in this control.

- Identity Proof Version 2 control if the RA verified the identity proof in this control.

- Full PKI Request if the RA performed an out-of-band identity proof for this request. The request SHOULD NOT contain either Identity Proof control.

- Simple PKI Request if the RA performed an out-of-band identity proof for this request.

The RA Identity Proof Witness control will frequently be associated with a Modify Certification Request control, which changes the name fields in the associated certification requests. This is because the RA knows the actual name to be assigned to the entity requesting the certificate, and the end-entity does not yet have the details of the name. (The association would be set up by the operator at the time the shared-secret was generated by the RA.)

When this control is placed in a message, it is RECOMMENDED that the Control Processed control be placed in the body sequence as well. Using the explicit new control, rather than implicitly relying on the Control Processed control is important due to the need to know explicitly which identity proofs have been performed. The new control also allows an RA to state that out-of-band identity proofs have been performed.

When the identity proof is performed by an RA, the RA also MUST validate the linking between the identity proof and the name information wrapped inside of the key proof-of-possession.

6.21. Response Body Control

The Response Body Control is designed to enable an RA to inform an EE that there is an embedded response message that MUST be processed as part of the processing of this message. This control is designed to be used in a couple of different cases where an RA has done some additional processing for the certification request, e.g., as key generation. When an RA performs key generation on behalf of an EE, the RA MUST respond with both the original response message from the certificate issuer (containing the certificate issuance) as part of the response generated by the RA (containing the new key). Another case where this is useful is when the secret is shared between the RA and the EE (rather than between the CA and the EE) and the RA returns the Publish Trust Anchors control (to populate the correct trust points).

The Response Body Control is identified by the OID:

```
id-cmc-responseBody OBJECT IDENTIFIER ::= { id-cmc 37 }
```

The Response Body Control has the ASN.1 definition:

```
cmc-responseBody CMC-CONTROL ::= {
   BodyPartPath IDENTIFIED BY id-cmc-responseBody
}
```

cmc-responseBody is a CMC-CONTROL associating the object identifier id-cmc-responseBody with the type BodyPartPath. This object is omitted from the 1988 module. The object is added to the object set Cmc-Control-Set. The control is permitted to appear only in the control sequence of a PKIResponse. The control MUST NOT appear in the control sequence of a PKIData. It is expected that only an intermediary RA will use this control; a CA generally does not need the control as it is creating the original innermost message.

id-cmc-responseBody is the object identifier used to identify this CMC control.

BodyPartPath is the type structure associated with the control. The syntax of BodyPartPath is defined in Section 3.2.2. The path contains a sequence of body part identifiers leading to a cmsSequence item which contains a PKIResponse within it.

7. Other Attributes

There are a number of different locations where various types of attributes can be placed in either a CMC request or a CMC response message. These places include the attribute sequence of a PKCS #10 request, controls in CRMF Section 6 of [CRMF], and the various CMS attribute sequences.

7.1. Change Subject Name Attribute

The Client Name Change Request attribute is designed for a client to ask for a change in its name as part of a certification request. Because of security issues, this cannot be done in the simple way of just changing the requested subject name in the certificate template. The name in the certification request MUST match the name in the certificate used to verify the request, in order that identity and possession proofs are correctly applied.

The relevant ASN.1 for the Client Name Change Request attribute is as follows:

```
at-cmc-changeSubjectName ATTRIBUTE ::=
  { ChangeSubjectName IDENTIFIED BY id-cmc-changeSubjectName }

id-cmc-changeSubjectName OBJECT IDENTIFIER ::= { id-cmc 36 }

ChangeSubjectName ::= SEQUENCE {
  subject             Name OPTIONAL,
  subjectAlt          [1] GeneralNames OPTIONAL
}
(WITH COMPONENTS {..., subject PRESENT} |
 WITH COMPONENTS {..., subjectAlt PRESENT} )
```

The attribute is designed to be used as an ATTRIBUTE object. As such, the attribute is placed in one of the following two places:

The attributes field in a CertificationRequest.

The controls field of a CertRequest for a CRMF certification request.

The control is identified by the Object Identifier id-cmc-changeSubjectName.

The ASN.1 type associated with control is ChangeSubjectName. The fields of the structure are configured as follows:

subject contains the requested subject name for the new certificate.

subjectAlt contains the requested subject alternative name for the new certificate.

At least one of the fields in the sequence MUST be present when encoding the structure.

When the CA processes this attribute in a certification request, it will do the following:

1. If present, the subject field is copied to the name field of the template. If the subject field is absent, the name field of the template will be set to a empty sequence.

2. If present, the subjectAlt field is used as the content of a SubjectAltName extension in the certificate. If the subjectAlt field is absent, the subjectAltName extension is removed from the certificate template.

8. Registration Authorities

This specification permits the use of RAs. An RA sits between the EE and the CA. From the EE's perspective, the RA appears to be the CA, and from the server, the RA appears to be a client. RAs receive the PKI Requests, perform local processing and then forward them onto CAs. Some of the types of local processing that an RA can perform include:

* Batching multiple PKI Requests together,

* Performing challenge/response POP proofs,

* Adding private or standardized certificate extensions to all certification requests,

* Archiving private key material,

* Routing requests to different CAs.

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

When an RA adds a new wrapping layer to a PKI Request, it creates a new PKIData. The new layer contains any controls required (for example, if the RA does the POP proof for an encryption key or the Add Extension control to modify a PKI Request) and the client PKI Request. The client PKI Request is placed in the cmsSequence if it is a Full PKI Request and in the reqSequence if it is a Simple PKI Request. If an RA is batching multiple client PKI Requests together, then each client PKI Request is placed into the appropriate location in the RA's PKIData object along with all relevant controls.

If multiple RAs are in the path between the EE and the CA, this will lead to multiple wrapping layers on the request.

In processing a PKI Request, an RA MUST NOT alter any certification requests (PKCS #10 or CRMF) as any alteration would invalidate the signature on the certification request and thus the POP for the private key.

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

```
SignedData (by RA)
   PKIData
     controlSequence
       RA added control statements
     reqSequence
       Zero or more Simple PKI Requests from clients
      cmsSequence
           Zero or more Full PKI Requests from clients
             SignedData (signed by client)
             PKIData
```

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

8.1. Encryption Removal

There are two cases that require an RA to remove or change encryption in a PKI Request. In the first case, the encryption was applied for the purposes of protecting the entire PKI Request from unauthorized entities. If the CA does not have a Recipient Info entry in the encryption layer, the RA MUST remove the encryption layer. The RA 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 RA 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 RA. If the signing layer provided by the EE needs to also be removed, the RA can also remove this layer.

8.2. Signature Layer Removal

Only two instances exist where an RA should remove a signature layer on a Full PKI Request: if an encryption layer needs to be modified within the request, or if a CA will not accept secondary delegation (i.e., multiple RA signatures). In all other situations, RAs SHOULD NOT remove a signing layer from a PKI Request.

If an RA removes a signing layer from a PKI Request, 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 RA.

9. Certificate Requirements

Certificates for servers used in the CMC protocol SHOULD conform to the profile defined in [PKIXCERT]. This document defines some additional items that MAY appear in CMC server certificates. Section 9.1 defines some additional values for the Extended Key Usage extension. Section 9.2 defines a new Subject Information Access value that allows for a CMC certificate to publish information on how to contact the services it provides.

9.1. Extended Key Usage

The Extended Key Usage (EKU) extension is used to restrict the use of a certificate to specific applications. We define three different EKUs in this document. The ASN.1 to define these EKUs is:

```
id-kp-cmcCA      OBJECT IDENTIFIER ::= { id-kp 27 }
id-kp-cmcRA      OBJECT IDENTIFIER ::= { id-kp 28 }
id-kp-cmcArchive OBJECT IDENTIFIER ::= { id-kp 29 }
```

The usage description for each of the EKUs is as follows:

CMC Certification Authorities are identified by the id-kp-cmcCA extended key usage. The certificate may be the same as or different than the CA certificate. If a different certificate is used, the certificates containing the id-kp-cmcCA extended key usage SHOULD have the same name as the certificate used for issuing the certificates. (Using a separate key pair for CMC protocol operations and for issuing certificates and CRLs decreases the number of operations for which the private key used to sign certificates and CRLs would be used.)

CMC Registration Authorities are identified by the id-kp-cmcRA extended key usage. This usage is placed into RA certificates.

CMC Archive Servers are identified by the id-kp-cmcArchive extended key usage. CMC Archive Servers and the associated protocol are to be defined in a future document.

9.2. Subject Information Access

The subject information access extension indicates how to access information and services for the subject of the certificate. We define a new value for use in this extension, to identify the different locations that CMC services will be available. If this value is placed in a certificate, an appropriate extended key usage defined in Section 9.1 MUST be included in the certificate as well.

The id-ad-cmc OID is used when the subject offers certification services using the CMC protocol. If the CMC services are available via HTTP or FTP, accessLocation MUST be a uniformResourceIdentifier. If the CMC services are available via electronic mail, accessLocation MUST be an rfc822Name. If CMC services are available using TCP/IP, the dNSName or iPAddress name forms MUST be used. Since the GeneralName data structure does not permit the inclusion of a port number, in the absence of other external configuration information, the value of 5318 should be used. (The port registration is in Section 3.2) The semantics of other name forms of accessLocation (when accessMethod is id-ad-cmc) are not defined by this specification.

The ASN.1 type for this extension is GeneralName see Section 4.2.1.8 of [PKIXCERT].

```
id-ad-cmc OBJECT IDENTIFIER ::= { id-ad 12 }
```

10. Security Considerations

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 controls described in Section 6.6. Developers of state-constrained PKI clients are strongly encouraged to incorporate the use of these controls.

Extreme care needs to be taken when archiving a signing key. The holder of the archived key may have the ability to use the key to generate forged signatures. There are however reasons why a signing key should be archived. An archived CA signing key can be recovered in the event of failure to continue to produced CRLs following a disaster.

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]. Methods of avoiding the small subgroup attack can be found in [SMALL-GROUP]. CMC implementations ought to be aware of this attack when doing parameter validations.

When using a shared-secret for authentication purposes, the shared-secret should be generated using good random number techniques [RANDOM]. User selection of the secret allows for dictionary attacks to be mounted.

Extreme care must be used when processing the Publish Trust Anchors control. Incorrect processing can lead to the practice of slamming where an attacker changes the set of trusted anchors in order to weaken security.

One method of controlling the use of the Publish Trust Anchors control is as follows. The client needs to associate with each trust anchor accepted by the client the source of the trust anchor. Additionally, the client should associate with each trust anchor the types of messages for which the trust anchor is valid (i.e., is the trust anchor used for validating S/MIME messages, TLS, or CMC enrollment messages?).

When a new message is received with a Publish Trust Anchors control, the client would accept the set of new trust anchors for specific applications only if the signature validates, the signer of the message has the required policy approval for updating the trust anchors, and local policy also would allow updating the trust anchors.

The CMS AuthenticatedData structure provides message integrity, it does not provide message authentication in all cases. When using MACs in this document the following restrictions need to be observed. All messages should be for a single entity. If two entities are placed in a single message, the entities can generate new messages that have a valid MAC and might be assumed to be from the original message sender. All entities that have access to the shared-secret can generate messages that will have a successful MAC validation. This means that care must be taken to keep this value secret. Whenever possible, the SignedData structure should be used in preference to the AuthenticatedData structure.

A number of controls such as the RA Identity Proof Witness control exist for an RA to either make assertions about or modify a certification request. Any upstream request processor, such as a CA, MUST verify that the RA is fully identified and authorized to make the assertion or modification it is claiming. If it is not identified or authorized, then any request MUST be rejected.

CMC servers, both RAs and CAs, need to perform due diligence in checking the contents of a certification request. At an absolute minimum, all fields should be checked to ensure that the policies of the CA/RA are correctly enforced. While all fields need to be checked, special care should be taken with names, name forms, algorithm choices, and algorithm parameters.

11. IANA Considerations

This document defines a number of control objects. These are identified by Object Identifiers (OIDs). The objects are defined from an arc delegated by IANA to the PKIX Working Group.

For the ASN.1 modules in Appendix A, IANA is requested to assign an OID for the module identifier (TBD1) with a Description of "id-mod-enrollMsgSyntax-2025" and an OID for the module identifier (TBD2) with a Description of "id-mod-pbkdf2-prfs-2025". The OIDs for the modules should be allocated in the "SMI Security for PKIX Module Identifier" registry (1.3.6.1.5.5.7.0).

12. References

12.1. Normative References

[CMS] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/rfc/rfc5652>.

[CMS-ALGS] Hoffman, P. and J. Schaad, "New ASN.1 Modules for Cryptographic Message Syntax (CMS) and S/MIME", RFC 5911, DOI 10.17487/RFC5911, June 2010, <https://www.rfc-editor.org/rfc/rfc5911>.

[CRMF] Schaad, J., "Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)", RFC 4211, DOI 10.17487/RFC4211, September 2005, <https://www.rfc-editor.org/rfc/rfc4211>.

[DH-POP] Schaad, J. and H. Prafullchandra, "Diffie-Hellman Proof-of-Possession Algorithms", RFC 6955, DOI 10.17487/RFC6955, May 2013, <https://www.rfc-editor.org/rfc/rfc6955>.

[HMAC-ALGS]
Schaad, J. and S. Turner, "Additional New ASN.1 Modules for the Cryptographic Message Syntax (CMS) and the Public Key Infrastructure Using X.509 (PKIX)", RFC 6268, DOI 10.17487/RFC6268, July 2011, <https://www.rfc-editor.org/rfc/rfc6268>.

[PKCS10] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <https://www.rfc-editor.org/rfc/rfc2986>.

[PKIX-ALGS]
Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <https://www.rfc-editor.org/rfc/rfc5912>.

[PKIXCERT] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

12.2. Informative References

[CMC-COMPL]
Mandel, J. and S. Turner, "Certificate Management Messages over CMS (CMC): Compliance Requirements", Work in Progress, Internet-Draft, draft-ietf-lamps-rfc5274bis-05, 1 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-rfc5274bis-05>.

[CMC-PROTv1]
Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <https://www.rfc-editor.org/rfc/rfc5272>.

[CMC-TRANS]
Mandel, J. and S. Turner, "Certificate Management over CMS (CMC): Transport Protocols", Work in Progress, Internet-Draft, draft-ietf-lamps-rfc5273bis-05, 1 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-rfc5273bis-05>.

[CMC-Updates]
Schaad, J., "Certificate Management over CMS (CMC) Updates", RFC 6402, DOI 10.17487/RFC6402, November 2011, <https://www.rfc-editor.org/rfc/rfc6402>.

[CMS-RI] Housley, R., Gray, J., and T. Okubo, "Using Key Encapsulation Mechanism (KEM) Algorithms in the Cryptographic Message Syntax (CMS)", RFC 9629, DOI 10.17487/RFC9629, August 2024, <https://www.rfc-editor.org/rfc/rfc9629>.

[erratum2063]
"RFC 5272 erratum 2063", 4 March 2010, <https://www.rfc-editor.org/errata/eid2063>.

[erratum2731]
"RFC 5272 erratum 2731", 23 February 2011, <https://www.rfc-editor.org/errata/eid2731>.

[erratum3943]
"RFC 6402 erratum 3943", 2 April 2014, <https://www.rfc-editor.org/errata/eid3943>.

[erratum4775]
"RFC 5272 erratum 4775", 11 August 2016, <https://www.rfc-editor.org/errata/eid4775>.

[erratum5931]
"RFC 6402 erratum 5931", 7 December 2019, <https://www.rfc-editor.org/errata/eid5931>.

[erratum6571]
"RFC 6402 erratum 6571", 4 May 2021, <https://www.rfc-editor.org/errata/eid6571>.

[erratum7379]
"RFC 5272 erratum 7379", 8 March 2023, <https://www.rfc-editor.org/errata/eid7379>.

[erratum7627]
"RFC 5272 erratum 7627", 4 September 2023,
<https://www.rfc-editor.org/errata/eid7627>.

[erratum7628]
"RFC 5272 erratum 7628", 4 September 2023,
<https://www.rfc-editor.org/errata/eid7628>.

[erratum7629]
"RFC 5272 erratum 7629", 4 September 2023,
<https://www.rfc-editor.org/errata/eid7629>.

[erratum8027]
"RFC 5272 erratum 8027", 11 July 2024,
<https://www.rfc-editor.org/errata/eid8027>.

[erratum8137]
"RFC 5272 erratum 8137", 12 October 2024,
<https://www.rfc-editor.org/errata/eid8137>.

[erratum8385]
"RFC 6402 erratum 8385", 18 April 2025,
<https://www.rfc-editor.org/errata/eid8385>.

[PASSWORD] Grassi, P., Garcia, M., and J. Fenton, "Digital identity guidelines: revision 3", National Institute of Standards and Technology, DOI 10.6028/nist.sp.800-63-3, June 2017, <https://doi.org/10.6028/nist.sp.800-63-3>.

[RANDOM] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/rfc/rfc4086>.

[RFC2797] Myers, M., Liu, X., Schaad, J., and J. Weinstein, "Certificate Management Messages over CMS", RFC 2797, DOI 10.17487/RFC2797, April 2000, <https://www.rfc-editor.org/rfc/rfc2797>.

[SMALL-GROUP]
Zuccherato, R., "Methods for Avoiding the "Small-Subgroup" Attacks on the Diffie-Hellman Key Agreement Method for S/MIME", RFC 2785, DOI 10.17487/RFC2785, March 2000, <https://www.rfc-editor.org/rfc/rfc2785>.

[X942] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, DOI 10.17487/RFC2631, June 1999, <https://www.rfc-editor.org/rfc/rfc2631>.

Appendix A. ASN.1 Modules

A.1. ASN.1 Module for CMC

```
<CODE BEGINS>
EnrollmentMessageSyntax-2025
    { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-enrollMsgSyntax-2025(TBD1) }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

  EXPORTS ALL;

  IMPORTS

  AttributeSet{}, Extension{}, EXTENSION, ATTRIBUTE
  FROM PKIX-CommonTypes-2009
      { iso(1) identified-organization(3) dod(6) internet(1) security(5)
      mechanisms(5) pkix(7) id-mod(0) id-mod-pkixCommon-02(57) }

  AlgorithmIdentifier{}, DIGEST-ALGORITHM, KEY-WRAP, KEY-DERIVATION,
      MAC-ALGORITHM, SIGNATURE-ALGORITHM, PUBLIC-KEY
  FROM AlgorithmInformation-2009
      {iso(1) identified-organization(3) dod(6) internet(1) security(5)
      mechanisms(5) pkix(7) id-mod(0)
      id-mod-algorithmInformation-02(58)}

  CertificateSerialNumber, GeneralName, CRLReason, ReasonFlags,
      CertExtensions, GeneralNames
  FROM PKIX1Implicit-2009
      { iso(1) identified-organization(3) dod(6) internet(1) security(5)
      mechanisms(5) pkix(7) id-mod(0) id-mod-pkix1-implicit-02(59) }

  Name, id-pkix, PublicKeyAlgorithms, SignatureAlgorithms, id-ad, id-kp
  FROM PKIX1Explicit-2009
      { iso(1) identified-organization(3) dod(6) internet(1) security(5)
        mechanisms(5) pkix(7) id-mod(0) id-mod-pkix1-explicit-02(51) }

  ContentInfo, IssuerAndSerialNumber, CONTENT-TYPE
  FROM CryptographicMessageSyntax-2010
    { iso(1) member-body(2) us(840) rsadsi(113549)
       pkcs(1) pkcs-9(9) smime(16) modules(0) id-mod-cms-2009(58) }

  CertReqMsg, PKIPublicationInfo, CertTemplate
  FROM PKIXCRMF-2009
```

```
        { iso(1) identified-organization(3) dod(6) internet(1) security(5)
          mechanisms(5) pkix(7) id-mod(0) id-mod-crmf2005-02(55) }

    mda-sha1
    FROM PKIXAlgs-2009
         { iso(1) identified-organization(3) dod(6)
           internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
           id-mod-pkix1-algorithms2008-02(56) }

    maca-hMAC-SHA1
    FROM CryptographicMessageSyntaxAlgorithms-2009
        { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
          smime(16) modules(0) id-mod-cmsalg-2001-02(37) }

    mda-sha256
    FROM PKIX1-PSS-OAEP-Algorithms-2009
         { iso(1) identified-organization(3) dod(6)
           internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
           id-mod-pkix1-rsa-pkalgs-02(54) }

    maca-hMAC-SHA256
    FROM HMAC-2010
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) mod(0) id-mod-hmac(74) }

    kda-PBKDF2
    FROM PBKDF2-PRFs-2025
        { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
          smime(16) modules(0) id-mod-pbkdf2-prfs-2025(TBD2) } ;

    --  CMS content types defined in this document

    CMC-ContentTypes CONTENT-TYPE ::= { ct-PKIData | ct-PKIResponse, ... }

    --  Signature Algorithms defined in this document

    SignatureAlgs SIGNATURE-ALGORITHM ::= { sa-noSignature }

    --  CMS Unsigned Attributes

    CMC-UnsignedAtts ATTRIBUTE ::= { aa-cmc-unsignedData }

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

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

    ct-PKIData CONTENT-TYPE ::=
```

```
        { TYPE PKIData IDENTIFIED BY id-cct-PKIData }

    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
    }

    BodyPartID ::= INTEGER(0..4294967295)

    TaggedAttribute ::= SEQUENCE {
        bodyPartID         BodyPartID,
        attrType           CMC-CONTROL.&id({Cmc-Control-Set}),
        attrValues         SET OF CMC-CONTROL.
                               &Type({Cmc-Control-Set}{@attrType})
    }

    Cmc-Control-Set CMC-CONTROL ::= {
        cmc-identityProof | cmc-dataReturn | cmc-regInfo |
        cmc-responseInfo | cmc-queryPending | cmc-popLinkRandom |
        cmc-popLinkWitness | cmc-identification | cmc-transactionId |
        cmc-senderNonce | cmc-recipientNonce | cmc-statusInfo |
        cmc-addExtensions | cmc-encryptedPOP | cmc-decryptedPOP |
        cmc-lraPOPWitness | cmc-getCert | cmc-getCRL |
        cmc-revokeRequest | cmc-confirmCertAcceptance |
        cmc-statusInfoV2 | cmc-trustedAnchors | cmc-authData |
        cmc-batchRequests | cmc-batchResponses | cmc-publishCert |
        cmc-modCertTemplate | cmc-controlProcessed |
        cmc-identityProofV2 | cmc-popLinkWitnessV2 |
        cmc-raIdentityWitness | cmc-responseBody, ... }

    OTHER-REQUEST ::= TYPE-IDENTIFIER

    --  We do not define any other requests in this document.
    --  Examples might be attribute certification requests.

    OtherRequests OTHER-REQUEST ::= {...}

    TaggedRequest ::= CHOICE {
        tcr               [0] TaggedCertificationRequest,
        crm               [1] CertReqMsg,
        orm               [2] SEQUENCE {
            bodyPartID            BodyPartID,
            requestMessageType    OTHER-REQUEST.&id({OtherRequests}),
            requestMessageValue   OTHER-REQUEST.&Type({OtherRequests}
```

```
                                    {@.requestMessageType})
        }
    }

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

    AttributeList ATTRIBUTE ::= { at-extension-req |
        at-cmc-changeSubjectName, ... }

    CertificationRequest ::= SEQUENCE {
       certificationRequestInfo  SEQUENCE {
           version                   INTEGER,
           subject                   Name,
           subjectPublicKeyInfo      SEQUENCE {
               algorithm                 AlgorithmIdentifier{PUBLIC-KEY,
                                             {PublicKeyAlgorithms}},
               subjectPublicKey          BIT STRING
           },
           attributes                [0] IMPLICIT SET OF
                                         AttributeSet{{AttributeList}}
        },
        signatureAlgorithm        AlgorithmIdentifier
                                      {SIGNATURE-ALGORITHM,
                                          {SignatureAlgorithms}},
        signature                 BIT STRING
    }

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

    OTHER-MSG ::= TYPE-IDENTIFIER

    --  No other messages currently defined

    OtherMsgSet OTHER-MSG ::= {...}

    OtherMsg ::= SEQUENCE {
        bodyPartID        BodyPartID,
        otherMsgType      OTHER-MSG.&id({OtherMsgSet}),
        otherMsgValue     OTHER-MSG.&Type({OtherMsgSet}{@otherMsgType}) }

    --  This defines the response message in the protocol
```

```
ct-PKIResponse CONTENT-TYPE ::=
    { TYPE PKIResponse IDENTIFIED BY id-cct-PKIResponse }

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

ResponseBody ::= PKIResponse

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

CMC-CONTROL ::= TYPE-IDENTIFIER

-- The following controls have the type OCTET STRING

cmc-identityProof CMC-CONTROL ::=
    { OCTET STRING IDENTIFIED BY id-cmc-identityProof }

id-cmc-identityProof OBJECT IDENTIFIER ::= { id-cmc 3 }

cmc-dataReturn CMC-CONTROL ::=
    { OCTET STRING IDENTIFIED BY id-cmc-dataReturn }

id-cmc-dataReturn OBJECT IDENTIFIER ::= { id-cmc 4 }

cmc-regInfo CMC-CONTROL ::=
    { OCTET STRING IDENTIFIED BY id-cmc-regInfo }

id-cmc-regInfo OBJECT IDENTIFIER ::= { id-cmc 18 }

cmc-responseInfo CMC-CONTROL ::=
    { OCTET STRING IDENTIFIED BY id-cmc-responseInfo }

id-cmc-responseInfo OBJECT IDENTIFIER ::= { id-cmc 19 }

cmc-queryPending CMC-CONTROL ::=
    { OCTET STRING IDENTIFIED BY id-cmc-queryPending }

id-cmc-queryPending OBJECT IDENTIFIER ::= { id-cmc 21 }

cmc-popLinkRandom CMC-CONTROL ::=
    { OCTET STRING IDENTIFIED BY id-cmc-popLinkRandom }

id-cmc-popLinkRandom OBJECT IDENTIFIER ::= { id-cmc 22 }

cmc-popLinkWitness CMC-CONTROL ::=
```

```
        { OCTET STRING IDENTIFIED BY id-cmc-popLinkWitness }

    id-cmc-popLinkWitness OBJECT IDENTIFIER ::= { id-cmc 23 }

    -- The following controls have the type UTF8String

    cmc-identification CMC-CONTROL ::=
        { UTF8String IDENTIFIED BY id-cmc-identification }

    id-cmc-identification OBJECT IDENTIFIER ::= { id-cmc 2 }

    -- The following controls have the type INTEGER

    cmc-transactionId CMC-CONTROL ::=
        { INTEGER IDENTIFIED BY id-cmc-transactionId }

    id-cmc-transactionId OBJECT IDENTIFIER ::= { id-cmc 5 }

    -- The following controls have the type OCTET STRING

    cmc-senderNonce CMC-CONTROL ::=
        { OCTET STRING IDENTIFIED BY id-cmc-senderNonce }

    id-cmc-senderNonce OBJECT IDENTIFIER ::= { id-cmc 6 }

    cmc-recipientNonce CMC-CONTROL ::=
        { OCTET STRING IDENTIFIED BY id-cmc-recipientNonce }

    id-cmc-recipientNonce OBJECT IDENTIFIER ::= { id-cmc 7 }

    -- Used to return status in a response

    cmc-statusInfo CMC-CONTROL ::=
        { CMCStatusInfo IDENTIFIED BY id-cmc-statusInfo }

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

    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 ::= INTEGER {
        success         (0),
        failed          (2),
        pending         (3),
        noSupport       (4),
        confirmRequired (5),
        popRequired     (6),
        partial         (7)
    }

    CMCFailInfo ::= INTEGER {
        badAlg          (0),
        badMessageCheck (1),
        badRequest      (2),
        badTime         (3),
        badCertId       (4),
        unsuportedExt   (5),
        mustArchiveKeys (6),
        badIdentity     (7),
        popRequired     (8),
        popFailed       (9),
        noKeyReuse      (10),
        internalCAError (11),
        tryLater        (12),
        authDataFail    (13)
    }

    -- Used for RAs to add extensions to certification requests

    cmc-addExtensions CMC-CONTROL ::=
        { AddExtensions IDENTIFIED BY id-cmc-addExtensions }

    id-cmc-addExtensions OBJECT IDENTIFIER ::= { id-cmc 8 }

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

    cmc-encryptedPOP CMC-CONTROL ::=
        { EncryptedPOP IDENTIFIED BY id-cmc-encryptedPOP }

    cmc-decryptedPOP CMC-CONTROL ::=
```

```
        { DecryptedPOP IDENTIFIED BY id-cmc-decryptedPOP }

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

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

    EncryptedPOP ::= SEQUENCE {
        request       TaggedRequest,
        cms             ContentInfo,
        thePOPAlgID     AlgorithmIdentifier{MAC-ALGORITHM, {POPAlgs}},
        witnessAlgID    AlgorithmIdentifier{DIGEST-ALGORITHM,
                            {WitnessAlgs}},
        witness         OCTET STRING
    }

    POPAlgs MAC-ALGORITHM ::= { maca-hMAC-SHA1 | maca-hMAC-SHA256, ... }

    WitnessAlgs DIGEST-ALGORITHM ::= { mda-sha1 | mda-sha256, ... }

    DecryptedPOP ::= SEQUENCE {
        bodyPartID      BodyPartID,
        thePOPAlgID     AlgorithmIdentifier{MAC-ALGORITHM, {POPAlgs}},
        thePOP          OCTET STRING
    }

    cmc-lraPOPWitness CMC-CONTROL ::=
        { LraPopWitness IDENTIFIED BY id-cmc-lraPOPWitness }

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

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

    cmc-getCert CMC-CONTROL ::=
        { GetCert IDENTIFIED BY id-cmc-getCert }

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

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

    cmc-getCRL CMC-CONTROL ::=
        { GetCRL IDENTIFIED BY id-cmc-getCRL }

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

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

cmc-revokeRequest CMC-CONTROL ::=
    { RevokeRequest IDENTIFIED BY id-cmc-revokeRequest}

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

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

cmc-confirmCertAcceptance CMC-CONTROL ::=
    { CMCCertId IDENTIFIED BY id-cmc-confirmCertAcceptance }

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

CMCCertId ::= IssuerAndSerialNumber

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

at-extension-req ATTRIBUTE ::=
    { TYPE ExtensionReq IDENTIFIED BY id-ExtensionReq }

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

ExtensionReq ::= SEQUENCE SIZE (1..MAX) OF
    Extension{{CertExtensions}}

-- The following allows Diffie-Hellman Certification Request
-- Messages to be well-formed

sa-noSignature SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-noSignature
    VALUE NoSignatureValue
    PARAMS TYPE NULL ARE required
    HASHES { mda-sha1 }
}
```

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

NoSignatureValue ::= OCTET STRING

--  Unauthenticated attribute to carry removable data.

id-aa OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
    rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) }

aa-cmc-unsignedData ATTRIBUTE ::=
    { TYPE CMCUnsignedData IDENTIFIED BY id-aa-cmc-unsignedData }

id-aa-cmc-unsignedData OBJECT IDENTIFIER ::= { id-aa 34 }

CMCUnsignedData ::= SEQUENCE {
    bodyPartPath        BodyPartPath,
    identifier          TYPE-IDENTIFIER.&id,
    content             TYPE-IDENTIFIER.&Type
}

--  Replaces CMC Status Info
--

cmc-statusInfoV2 CMC-CONTROL ::=
    { CMCStatusInfoV2 IDENTIFIED BY id-cmc-statusInfoV2 }

id-cmc-statusInfoV2 OBJECT IDENTIFIER ::= { id-cmc 25 }

EXTENDED-FAILURE-INFO ::= TYPE-IDENTIFIER

ExtendedFailures EXTENDED-FAILURE-INFO ::= {...}

CMCStatusInfoV2 ::= SEQUENCE {
   cMCStatus             CMCStatus,
   bodyList              SEQUENCE SIZE (1..MAX) OF
                                 BodyPartReference,
   statusString          UTF8String OPTIONAL,
   otherInfo             CHOICE {
       failInfo              CMCFailInfo,
       pendInfo              PendInfo,
       extendedFailInfo      [1] SEQUENCE {
          failInfoOID           TYPE-IDENTIFIER.&id
                                    ({ExtendedFailures}),
          failInfoValue         TYPE-IDENTIFIER.&Type
                                    ({ExtendedFailures}
                                        {@.failInfoOID})
       }
    } OPTIONAL
```

```
}

BodyPartReference ::= CHOICE {
   bodyPartID           BodyPartID,
   bodyPartPath         BodyPartPath
}

BodyPartPath ::= SEQUENCE SIZE (1..MAX) OF BodyPartID

--  Allow for distribution of trust anchors

cmc-trustedAnchors CMC-CONTROL ::=
    { PublishTrustAnchors IDENTIFIED BY id-cmc-trustedAnchors }

id-cmc-trustedAnchors OBJECT IDENTIFIER ::= { id-cmc 26 }

PublishTrustAnchors ::= SEQUENCE {
    seqNumber      INTEGER,
    hashAlgorithm  AlgorithmIdentifier{DIGEST-ALGORITHM,
                       {HashAlgorithms}},
    anchorHashes   SEQUENCE OF OCTET STRING
}

HashAlgorithms DIGEST-ALGORITHM ::= {
   mda-sha1 | mda-sha256, ...
}

cmc-authData CMC-CONTROL ::=
    { AuthPublish IDENTIFIED BY id-cmc-authData }

id-cmc-authData OBJECT IDENTIFIER ::= { id-cmc 27 }

AuthPublish ::= BodyPartID

--   These two items use BodyPartList

cmc-batchRequests CMC-CONTROL ::=
    { BodyPartList IDENTIFIED BY id-cmc-batchRequests }

id-cmc-batchRequests OBJECT IDENTIFIER ::= { id-cmc 28 }

cmc-batchResponses CMC-CONTROL ::=
    { BodyPartList IDENTIFIED BY id-cmc-batchResponses }

id-cmc-batchResponses OBJECT IDENTIFIER ::= { id-cmc 29 }

BodyPartList ::= SEQUENCE SIZE (1..MAX) OF BodyPartID
```

```
    cmc-publishCert CMC-CONTROL ::=
        { CMCPublicationInfo IDENTIFIED BY id-cmc-publishCert }

    id-cmc-publishCert OBJECT IDENTIFIER ::= { id-cmc 30 }

    CMCPublicationInfo ::= SEQUENCE {
        hashAlg        AlgorithmIdentifier{DIGEST-ALGORITHM,
                             {HashAlgorithms}},
        certHashes     SEQUENCE OF OCTET STRING,
        pubInfo        PKIPublicationInfo
    }

    cmc-modCertTemplate CMC-CONTROL ::=
        { ModCertTemplate IDENTIFIED BY id-cmc-modCertTemplate }

    id-cmc-modCertTemplate OBJECT IDENTIFIER ::= { id-cmc 31 }

    ModCertTemplate ::= SEQUENCE {
        pkiDataReference            BodyPartPath,
        certReferences              BodyPartList,
        replace                     BOOLEAN DEFAULT TRUE,
        certTemplate                CertTemplate
    }

    -- Inform follow-on servers that one or more controls have
    -- already been processed

    cmc-controlProcessed CMC-CONTROL ::=
        { ControlsProcessed IDENTIFIED BY id-cmc-controlProcessed }

    id-cmc-controlProcessed OBJECT IDENTIFIER ::= { id-cmc 32 }

    ControlsProcessed ::= SEQUENCE {
        bodyList              SEQUENCE SIZE(1..MAX) OF BodyPartReference
    }

    --  Identity Proof control w/ algorithm agility

    cmc-identityProofV2 CMC-CONTROL ::=
        { IdentityProofV2 IDENTIFIED BY id-cmc-identityProofV2 }

    id-cmc-identityProofV2 OBJECT IDENTIFIER ::= { id-cmc 33 }

    IdentityProofV2 ::= SEQUENCE {
        proofAlgID       AlgorithmIdentifier{DIGEST-ALGORITHM,
                             {WitnessAlgs}},
        macAlgId         AlgorithmIdentifier{MAC-ALGORITHM, {POPAlgs}},
        witness          OCTET STRING
```

```
}

cmc-popLinkWitnessV2 CMC-CONTROL ::=
    { PopLinkWitnessV2 IDENTIFIED BY id-cmc-popLinkWitnessV2 }

id-cmc-popLinkWitnessV2 OBJECT IDENTIFIER ::= { id-cmc 34 }

PopLinkWitnessV2 ::= SEQUENCE {
    keyGenAlgorithm   AlgorithmIdentifier{KEY-DERIVATION,
                          {KeyDevAlgs}},
    macAlgorithm      AlgorithmIdentifier{MAC-ALGORITHM, {POPAlgs}},
    witness           OCTET STRING
}

KeyDevAlgs KEY-DERIVATION ::= { kda-PBKDF2, ... }

cmc-raIdentityWitness CMC-CONTROL ::=
   { BodyPartPath IDENTIFIED BY id-cmc-raIdentityWitness }

id-cmc-raIdentityWitness OBJECT IDENTIFIER ::= {id-cmc 35}

--
--  Allow for an End-Entity to request a change in name.
--  This item is added to RegControlSet in CRMF.
--
at-cmc-changeSubjectName ATTRIBUTE ::=
   { TYPE ChangeSubjectName IDENTIFIED BY id-cmc-changeSubjectName }

id-cmc-changeSubjectName OBJECT IDENTIFIER ::= { id-cmc 36 }

ChangeSubjectName ::= SEQUENCE {
    subject             Name OPTIONAL,
    subjectAlt          [1] GeneralNames OPTIONAL
}
(WITH COMPONENTS {..., subject PRESENT} |
 WITH COMPONENTS {..., subjectAlt PRESENT} )

--
--  Embedded response from a third party for processing
--

cmc-responseBody CMC-CONTROL ::= {
   BodyPartPath IDENTIFIED BY id-cmc-responseBody
}

id-cmc-responseBody OBJECT IDENTIFIER ::= { id-cmc 37 }

--
```

```
     --  Key purpose identifiers are in the Extended Key Usage extension
     --

     id-kp-cmcCA OBJECT IDENTIFIER ::= { id-kp 27 }
     id-kp-cmcRA OBJECT IDENTIFIER ::= { id-kp 28 }
     id-kp-cmcArchive OBJECT IDENTIFIER ::= { id-kp 29 }

     --
     --  Subject Information Access identifier
     --

     id-ad-cmc OBJECT IDENTIFIER ::= { id-ad 12 }

   END
   <CODE ENDS>
```

A.2. ASN.1 Module for PBKDF2 PRFs

```
   <CODE BEGINS>
   PBKDF2-PRFs-2025
     { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
       pkcs-9(9) smime(16) modules(0) id-mod-pbkdf2-prfs-2025(TBD2) }

     DEFINITIONS IMPLICIT TAGS ::= BEGIN

     IMPORTS

     ALGORITHM, AlgorithmIdentifier{}, KEY-DERIVATION
     FROM AlgorithmInformation-2009 -- From [PKIX-ALGS]
         { iso(1) identified-organization(3) dod(6) internet(1)
           security(5) mechanisms(5) pkix(7) id-mod(0)
           id-mod-algorithmInformation-02(58) }

     hMAC-SHA1, alg-hMAC-SHA1, id-PBKDF2
     FROM CryptographicMessageSyntaxAlgorithms-2009 -- From [RFC5911]
         { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
            smime(16) modules(0) id-mod-cmsalg-2001-02(37) }

     id-hmacWithSHA224, id-hmacWithSHA256,
     id-hmacWithSHA384, id-hmacWithSHA512
     FROM HMAC-2010 -- From [HMAC-ALGS]
         { iso(1) identified-organization(3) dod(6) internet(1)
           security(5) mechanisms(5) pkix(7) mod(0) id-mod-hmac(74) } ;

     -- Base OID for algorithms --

     rsadsi OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
         rsadsi(113549) }
```

```
    digestAlgorithm OBJECT IDENTIFIER ::= { rsadsi 2 }

    id-hmacWithSHA512-224 OBJECT IDENTIFIER ::= { digestAlgorithm 12 }

    id-hmacWithSHA512-256 OBJECT IDENTIFIER ::= { digestAlgorithm 13 }

    -- PBKF2-PRFs --

    PBKDF2-PRFs ALGORITHM ::= {
        alg-hMAC-SHA1 |
        alg-hMAC-SHA224 | alg-hMAC-SHA256 |
        alg-hMAC-SHA384 | alg-hMAC-SHA512 |
        alg-hMAC-SHA512-224 | alg-hMAC-SHA512-256, ... }

    PBKDF2-PRFsAlgorithmIdentifier ::=
        AlgorithmIdentifier{ ALGORITHM, {PBKDF2-PRFs} }

    alg-hMAC-SHA224 ALGORITHM ::= { IDENTIFIER id-hmacWithSHA224
        PARAMS TYPE NULL ARE preferredAbsent }

    alg-hMAC-SHA256 ALGORITHM ::= { IDENTIFIER id-hmacWithSHA256
        PARAMS TYPE NULL ARE preferredAbsent }

    alg-hMAC-SHA384 ALGORITHM ::= { IDENTIFIER id-hmacWithSHA384
        PARAMS TYPE NULL ARE preferredAbsent }

    alg-hMAC-SHA512 ALGORITHM ::= { IDENTIFIER id-hmacWithSHA512
        PARAMS TYPE NULL ARE preferredAbsent }

    alg-hMAC-SHA512-224 ALGORITHM ::= { IDENTIFIER id-hmacWithSHA512-224
        PARAMS TYPE NULL ARE preferredAbsent }

    alg-hMAC-SHA512-256 ALGORITHM ::= { IDENTIFIER id-hmacWithSHA512-256
        PARAMS TYPE NULL ARE preferredAbsent }

    -- PBKF2-SaltSources --

    PBKDF2-SaltSources ALGORITHM ::= { ... }

    PBKDF2-SaltSourcesAlgorithmIdentifier ::=
        AlgorithmIdentifier {ALGORITHM, {PBKDF2-SaltSources} }

    -- PBKF2-params --

    PBKDF2-params ::= SEQUENCE {
        salt CHOICE {
            specified OCTET STRING,
            otherSource PBKDF2-SaltSourcesAlgorithmIdentifier },
```

```
        iterationCount INTEGER (1..MAX),
        keyLength INTEGER (1..MAX) OPTIONAL,
        prf PBKDF2-PRFsAlgorithmIdentifier DEFAULT defaultPBKDF2 }

    defaultPBKDF2 PBKDF2-PRFsAlgorithmIdentifier ::=
        { algorithm alg-hMAC-SHA1.&id, parameters NULL:NULL }

    -- Key Derivation Algorithms --

    KeyDerivationAlgs KEY-DERIVATION ::= { kda-PBKDF2, ... }

    kda-PBKDF2 KEY-DERIVATION ::= {
        IDENTIFIER id-PBKDF2
        PARAMS TYPE PBKDF2-params ARE required
        -- No S/MIME caps defined
    }

  END
  <CODE ENDS>
```

Appendix B. Enrollment Message Flows

This section is informational. The purpose of this section is to present, in an abstracted version, the messages that would flow between the client and server for several different common cases.

B.1. Request of a Signing Certificate

This section looks at the messages that would flow in the event that an enrollment is occurring for a signing-only key. If the certificate was designed for both signing and encryption, the only difference would be the key usage extension in the certification request.

Message from client to server:

```
      ContentInfo.contentType = id-signedData
      ContentInfo.content
        SignedData.encapContentInfo
          eContentType = id-cct-PKIData
          eContent
            controlSequence
              {102, id-cmc-identityProof, computed value}
              {103, id-cmc-senderNonce, 10001}
            reqSequence
              crm
                certReq
                  certReqId = 201
                  certTemplate
                    subject = My Proposed DN
                    publicKey = My Public Key
                    extensions
                      {id-ce-subjectKeyIdentifier, 1000}
                      {id-ce-keyUsage, digitalSignature}
        SignedData.SignerInfos
          SignerInfo
            sid.subjectKeyIdentifier = 1000
```

Response from server to client:

```
      ContentInfo.contentType = id-signedData
      ContentInfo.content
        SignedData.encapContentInfo
          eContentType = id-cct-PKIResponse
          eContent
            controlSequence
              {102, id-cmc-statusInfoV2, {success, 201}}
              {103, id-cmc-senderNonce, 10005}
              {104, id-cmc-recipientNonce, 10001}
        certificates
          Newly issued certificate
          Other certificates
        SignedData.SignerInfos
          Signed by CA
```

B.2. Single Certification Request, But Modified by RA

This section looks at the messages that would flow in the event that an enrollment has one RA in the middle of the data flow. That RA will modify the certification request before passing it on to the CA.

Message from client to RA:

```
      ContentInfo.contentType = id-signedData
      ContentInfo.content
        SignedData.encapContentInfo
          eContentType = id-cct-PKIData
          eContent
            controlSequence
              {102, id-cmc-identityProof, computed value}
              {103, id-cmc-senderNonce, 10001}
            reqSequence
              crm
                certReq
                  certReqId = 201
                  certTemplate
                    subject = My Proposed DN
                    publicKey = My Public Key
                    extensions
                      {id-ce-subjectKeyIdentifier, 1000}
                      {id-ce-keyUsage, digitalSignature}
        SignedData.SignerInfos
          SignerInfo
            sid.subjectKeyIdentifier = 1000
```

Message from RA to CA:

```
      ContentInfo.contentType = id-signedData
      ContentInfo.content
        SignedData.encapContentInfo
          eContentType = id-cct-PKIData
          eContent
            controlSequence
              { 102, id-cmc-batchRequests, { 1, 2} }
              { 103, id-cmc-addExtensions,
                { {1, 201, {id-ce-certificatePolicies, anyPolicy}}
                  {1, 201, {id-ce-subjectAltName, {extension data}}
                  {2, XXX, {id-ce-subjectAltName, {extension data}}}
                        The Value XXX is not known here; it would
                        reference into the second client request,
                        which is not displayed above.
            cmsSequence
              { 1, <Message from client to RA #1> }
              { 2, <Message from client to RA #2> }
        SignedData.SignerInfos
          SignerInfo
            sid = RA key.
```

Response from CA to RA:

```
      ContentInfo.contentType = id-signedData
      ContentInfo.content
        SignedData.encapContentInfo
          eContentType = id-cct-PKIResponse
          eContent
            controlSequence
              {102, id-cmc-BatchResponse, {999, 998}}

              {103, id-cmc-statusInfoV2, {failed, 2, badIdentity}}
            cmsSequence
              { bodyPartID = 999
                contentInfo
                  ContentInfo.contentType = id-signedData
                  ContentInfo.content
                    SignedData.encapContentInfo
                      eContentType = id-cct-PKIResponse
                      eContent
                        controlSequence
                         {102, id-cmc-statusInfoV2, {success, 201}}
                    certificates
                      Newly issued certificate
                      Other certificates
                    SignedData.SignerInfos
                      Signed by CA
              }
              { bodyPartID = 998,
                contentInfo
                  ContentInfo.contentType = id-signedData
                  ContentInfo.content
                    SignedData.encapContentInfo
                      eContentType = id-cct-PKIResponse
                      eContent
                        controlSequence
                          {102, id-cmc-statusInfoV2, {failure, badAlg}}
                    certificates
                      Newly issued certificate
                      Other certificates
                    SignedData.SignerInfos
                      Signed by CA
              }
            SignedData.SignerInfos
              Signed by CA
```

Response from RA to client:

```
ContentInfo.contentType = id-signedData
ContentInfo.content
  SignedData.encapContentInfo
    eContentType = id-cct-PKIResponse
    eContent
      controlSequence
        {102, id-cmc-statusInfoV2, {success, 201}}
  certificates
    Newly issued certificate
    Other certificates
  SignedData.SignerInfos
    Signed by CA
```

B.3. Direct POP for an RSA or KEM Certificate

This section looks at the messages that would flow in the event that an enrollment is done for an encryption-only certificate using a direct POP method; the example below shows. For simplicity, it is assumed that the certification requester already has a signature certificate.

Message #1 from client to server:

```
     ContentInfo.contentType = id-signedData
     ContentInfo.content
       SignedData.encapContentInfo
         eContentType = id-cct-PKIData
         eContent
           controlSequence
             {102, id-cmc-transactionId, 10132985123483401}
             {103, id-cmc-senderNonce, 10001}
             {104, id-cmc-dataReturn, <packet of binary data identifying
                                       where the key in question is.>}
           reqSequence
             crm
               certReq
                 certReqId = 201
                 certTemplate
                   subject = <My DN>
                   publicKey = My Public Key
                   extensions
                     {id-ce-keyUsage, keyEncipherment}
                     {id-ce-subjectPublicKeyIdentifier, 1000}
               popo
                 keyEncipherment
                   subsequentMessage = challengeResp
       SignedData.SignerInfos
         SignerInfo
           Signed by requester's signing cert
```

Response #1 from server to client:

```
   ContentInfo.contentType = id-signedData
   ContentInfo.content
     SignedData.encapContentInfo
       eContentType = id-cct-PKIResponse
       eContent
         controlSequence
           {101, id-cmc-statusInfoV2, {failed, 201, popRequired}}
           {102, id-cmc-transactionId, 10132985123483401}
           {103, id-cmc-senderNonce, 10005}
           {104, id-cmc-recipientNonce, 10001}
           {105, id-cmc-encryptedPOP, {
              request {
                crm
                  certReq
                    certReqId = 201
                     certTemplate
                       subject = <My DN>
                       publicKey = My Public Key
                       extensions
                         {id-ce-keyUsage, keyEncipherment}
                         {id-ce-subjectPublicKeyIdentifier, 1000}
                   popo
                     keyEncipherment
                       subsequentMessage = challengeResp
              }
              cms
                contentType = id-envelopedData
                content
                  recipientInfos.rid.issuerSerialNumber = <NULL-DN, 201>
                  encryptedContentInfo
                    eContentType = id-data
                    eContent = <Encrypted value of 'y' from Section 6.7>
              thePOPAlgID = HMAC-SHA256
              witnessAlgID = SHA-256
              witness <hashed value of 'y' from Section 6.7>}}
           {106, id-cmc-dataReturn, <packet of binary data identifying
                                     where the key in question is.>}
     certificates
       Other certificates
         (optional - related to this message's SignedData)
     SignedData.SignerInfos
       Signed by CA

   Message #2 from client to server:
```

```
     ContentInfo.contentType = id-signedData
     ContentInfo.content
       SignedData.encapContentInfo
         eContentType = id-cct-PKIData
         eContent
           controlSequence
             {102, id-cmc-transactionId, 10132985123483401}
             {103, id-cmc-senderNonce, 100101}
             {104, id-cmc-dataReturn, <packet of binary data identifying
                                       where the key in question is.>}
             {105, id-cmc-recipientNonce, 10005}
             {107, id-cmc-decryptedPOP, {
               bodyPartID 201,
               thePOPAlgID HMAC-SHA256,
               thePOP <HMAC computed value goes here>}}
           reqSequence
             crm
               certReq
                 certReqId = 201
                 certTemplate
                   subject = <My DN>
                   publicKey = My Public Key
                   extensions
                     {id-ce-keyUsage, keyEncipherment}
                     {id-ce-subjectKeyIdentifier, 1000}
               popo
                 keyEncipherment
                   subsequentMessage = challengeResp
       SignedData.SignerInfos
         SignerInfo
           Signed by requester's signing cert

   Response #2 from server to client:
```

```
ContentInfo.contentType = id-signedData
ContentInfo.content
  SignedData.encapContentInfo
    eContentType = id-cct-PKIResponse
    eContent
      controlSequence
        {101, id-cmc-transactionId, 10132985123483401}
        {102, id-cmc-statusInfoV2, {success, 201}}
        {103, id-cmc-senderNonce, 10019}
        {104, id-cmc-recipientNonce, 100101}
        {105, id-cmc-dataReturn, <packet of binary data identifying
                                  where the key in question is.>}
  certificates
    Newly issued certificate
    Other certificates
      (optional - related to this message's SignedData)
  SignedData.SignerInfos
    Signed by CA
```

B.4. Direct POP with No Signature Mechanism

This section looks at the messages that would flow in the event that an enrollment is done for an encryption-only certificate using a direct POP method. Instead of assuming that the certification requester already has a signing-only certificate as in Appendix B.3, here the No Signature mechanism from Appendix C.1, the public key is for a KEM, and the EnvelopedData uses the KEMRecipientInfo from [CMS-RI].

Message #1 from client to server:

```
     ContentInfo.contentType = id-signedData
     ContentInfo.content
       SignedData.encapContentInfo
         eContentType = id-cct-PKIData
         eContent
           controlSequence
             {102, id-cmc-transactionId, 10132985123483401}
             {103, id-cmc-senderNonce, 10001}
             {104, id-cmc-dataReturn, <packet of binary data identifying
                                       where the key in question is.>}
           reqSequence
             crm
               certReq
                 certReqId = 201
                 certTemplate
                   subject = < My DN >
                   publicKey = My Public Key
                   extensions
                     {id-ce-subjectPublicKeyIdentifier, 1000}
                     {id-ce-keyUsage, keyEncipherment}
               popo
                 keyEncipherment
                   subsequentMessage = challengeResp
       SignedData.SignerInfos
        SignerInfo
          sid = < subjectKeyIdentifier >
          signatureAlgorithm = id-alg-noSignature
```

Response #1 from server to client:

```
   ContentInfo.contentType = id-signedData
   ContentInfo.content
     SignedData.encapContentInfo
       eContentType = id-cct-PKIResponse
       eContent
         controlSequence
           {101, id-cmc-statusInfoV2, {failed, 201, popRequired}}
           {102, id-cmc-transactionId, 10132985123483401}
           {103, id-cmc-senderNonce, 10005}
           {104, id-cmc-recipientNonce, 10001}
           {105, id-cmc-encryptedPOP, {
              request {
                crm
                  certReq
                    certReqId = 201
                     certTemplate
                       subject = < My DN >
                       publicKey = My Public Key
                       extensions
                         {id-ce-keyUsage, keyEncipherment}
                         {id-ce-subjectPublicKeyIdentifier, 1000}
                   popo
                     keyEncipherment
                       subsequentMessage = challengeResp
              }
              cms
                contentType = id-envelopedData
                content < uses ori.KEMRecipientInfo >
                  recipientInfos.ori.rid.issuerSerialNumber =
                    <NULL-DN, 201>
                  encryptedContentInfo
                    eContentType = id-data
                    eContent = <Encrypted value of 'y' from Section 6.7>
       thePOPAlgID = KmacWithSHAKE128
       witnessAlgID = SHAKE128
       witness <hashed value of 'y' from Section 6.7>}}
           {106, id-cmc-dataReturn, <packet of binary data identifying
                                     where the key in question is.>}
     Certificates
       Other certificates
         (optional - related to this message's SignedData)
     SignedData.SignerInfos
       Signed by CA

   Message #2 from client to server:
```

```
     ContentInfo.contentType = id-signedData
     ContentInfo.content
       SignedData.encapContentInfo
         eContentType = id-cct-PKIData
         eContent
           controlSequence
             {102, id-cmc-transactionId, 10132985123483401}
             {103, id-cmc-senderNonce, 100101}
             {104, id-cmc-dataReturn, <packet of binary data identifying
                                       where the key in question is.>}
             {105, id-cmc-recipientNonce, 10005}
             {107, id-cmc-decryptedPOP, {
               bodyPartID 201,
               thePOPAlgID KmacWithSHAKE128,
               thePOP <KMAC computed value goes here>}}
           reqSequence
             crm
               certReq
                 certReqId = 201
                 certTemplate
                   subject = < My DN >
                   publicKey = My Public Key
                   extensions
                     {id-ce-keyUsage, keyEncipherment}
                     {id-ce-subjectPublicKeyIdentifier, 1000}
               popo
                 keyEncipherment
                   subsequentMessage = challengeResp
       SignedData.SignerInfos
         SignerInfo
           sid = < subjectKeyIdentifier >
           signatureAlgorithm = id-alg-noSignature
```

Response #2 from server to client:

```
ContentInfo.contentType = id-signedData
ContentInfo.content
  SignedData.encapContentInfo
    eContentType = id-cct-PKIResponse
    eContent
      controlSequence
        {101, id-cmc-transactionId, 10132985123483401}
        {102, id-cmc-statusInfoV2, {success, 201}}
        {103, id-cmc-senderNonce, 10019}
        {104, id-cmc-recipientNonce, 100101}
        {105, id-cmc-dataReturn, <packet of binary data identifying
                                  where the key in question is.>}
  certificates
    Newly issued certificate
    Other certificates
  SignedData.SignerInfos
    Signed by CA
```

Appendix C. Production of Diffie-Hellman Public Key Certification Requests

Part of a certification request is a signature over the request; DH and ECDH are key agreement algorithms and RSA-KEM and ML-KEM are key encapsulation mechanisms (KEM) and cannot be used to directly produce the required signature object. [DH-POP] provides three 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.

C.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 SHA-1 hash of the certification request. The hash value given by NoSignatureValue SHOULD be ignored. 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 6.7 MUST be used.

When the client generates the SignedData.SignerInfos.SignerInfo.sid field it has two choices: issuerAndSerialNumber or subjectKeyIdentifier. The client does not yet have a certificate and there cannot fill in the issuerAndSerialNumber and therefore MUST use the subjectKeyIdentifier choice.

Acknowledgments

Obviously, the authors of this version of the document would like to thank Jim Schaad and Michael Myers for their work on the previous version of this document.

The acknowledgment from the previous version of this document follows:

The authors and the PKIX Working Group are grateful for the participation of Xiaoyi Liu and Jeff Weinstein in helping to author the original versions of this document.

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.

Contributors

Jim Schaad
August Cellars

Michael Myers
TraceRoute Security, Inc.

Authors' Addresses

Joseph Mandel (editor)
AKAYLA, Inc.
Email: joe@akayla.com

Sean Turner (editor)
sn3rd
Email: sean@sn3rd.com