Workgroup: LAMPS Working Group
Internet-Draft: draft-ietf-lamps-rfc4210bis-06
Obsoletes: 4210 (if approved)
Updates: 5912 (if approved)
Published: 13 March 2023
Intended Status: Standards Track
Expires: 14 September 2023
Authors: H. Brockhaus (Siemens), D. von Oheimb (Siemens), M. Ounsworth (Entrust), J. Gray (Entrust)

Internet X.509 Public Key Infrastructure -- Certificate Management Protocol (CMP)

Abstract

This document describes the Internet X.509 Public Key Infrastructure (PKI) Certificate Management Protocol (CMP). Protocol messages are defined for X.509v3 certificate creation and management. CMP provides interactions between client systems and PKI components such as a Registration Authority (RA) and a Certification Authority (CA).

This document obsoletes RFC 4210 by including the updates specified by CMP Updates [RFCAAAA] Section 2 and Appendix A.2 maintaining backward compatibility with CMP version 2 wherever possible and obsoletes both documents. Updates to CMP version 2 are: improving crypto agility, extending the polling mechanism, adding new general message types, and adding extended key usages to identify special CMP server authorizations. Introducing version 3 to be used only for changes to the ASN.1 syntax, which are: support of EnvelopedData instead of EncryptedValue and hashAlg for indicating a hash AlgorithmIdentifier in certConf messages.

In addition to the changes specified in CMP Updates [RFCAAAA] this document adds support for management of KEM certificates.

Appendix F of this document updates the 2002 ASN.1 module in RFC 5912 Section 9.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[RFC Editor: please delete:

During IESG telechat the CMP Updates document was approved on condition that LAMPS provides a RFC4210bis document. Version -00 of this document shall be identical to RFC 4210 and version -01 incorporates the changes specified in CMP Updates Section 2 and Appendix A.2.

A history of changes is available in Appendix F of this document.

The authors of this document wish to thank Carlisle Adams, Stephen Farrell, Tomi Kause, and Tero Mononen, the original authors of RFC4210, for their work and invite them, next to further volunteers, to join the -bis activity as co-authors.

]

[RFC Editor:

Please perform the following substitution.

* RFCXXXX --> the assigned numerical RFC value for this draft

* RFCAAAA --> the assigned numerical RFC value for [I-D.ietf-lamps-cmp-updates]

  Add this RFC number to the list of obsoleted RFCs.

* RFCBBBB --> the assigned numerical RFC value for [I-D.ietf-lamps-lightweight-cmp-profile]

* RFCCCCC --> the assigned numerical RFC value for [I-D.ietf-lamps-cmp-algorithms]

* RFCDDDD --> the assigned numerical RFC value for [I-D.ietf-lamps-rfc6712bis]

* RFCEEEE --> the assigned numerical RFC value for [I-D.ietf-ace-cmpv2-coap-transport]

]

This document describes the Internet X.509 Public Key Infrastructure (PKI) Certificate Management Protocol (CMP). Protocol messages are defined for certificate creation and management. The term "certificate" in this document refers to an X.509v3 Certificate as defined in [ITU.X509.2000].

1.1. Changes Since RFC 2510

RFC 4210 [RFC4210] differs from RFC 2510 [RFC2510] in the following areas:

* The PKI management message profile section is split to two appendices: the required profile and the optional profile. Some of the formerly mandatory functionality is moved to the optional profile.

* The message confirmation mechanism has changed substantially.

* A new polling mechanism is introduced, deprecating the old polling method at the CMP transport level.

* The CMP transport protocol issues are handled in a separate document [I-D.ietf-lamps-rfc6712bis], thus the Transports section is removed.

* A new implicit confirmation method is introduced to reduce the number of protocol messages exchanged in a transaction.

* The new specification contains some less prominent protocol enhancements and improved explanatory text on several issues.

1.2. Changes Since RFC 4210

CMP Updates [RFCAAAA] and CMP Algorithms [RFCCCCC] updated RFC 4210 [RFC4210], supporting the PKI management operations specified in the Lightweight CMP Profile [RFCBBBB], in the following areas:

* Add new extended key usages for various CMP server types, e.g., registration authority and certification authority, to express the authorization of the certificate holder to act as the indicated type of PKI management entity.

* Extend the description of multiple protection to cover additional use cases, e.g., batch processing of messages.

* Use the type EnvelopedData as the preferred choice next to EncryptedValue to better support crypto agility in CMP.

  For reasons of completeness and consistency the type EncryptedValue has been exchanged in all occurrences. This includes the protection of centrally generated private keys, encryption of certificates, and protection of revocation passphrases. To properly differentiate the support of EnvelopedData instead of EncryptedValue, the CMP version 3 is introduced in case a transaction is supposed to use EnvelopedData.

  Note: According to RFC 4211 [RFC4211] Section 2.1. point 9 the use of the EncryptedValue structure has been deprecated in favor of the EnvelopedData structure. RFC 4211 [RFC4211] offers the EncryptedKey structure, a choice of EncryptedValue and EnvelopedData for migration to EnvelopedData.

* Offer an optional hashAlg field in CertStatus supporting case that a certificate needs to be confirmed that has a signature

algorithm that does not indicate a specific hash algorithm to use for computing the certHash.

* Add new general message types to request CA certificates, a root CA update, a certificate request template, or CRL updates.

* Extend the use of polling to p10cr, certConf, rr, genm, and error messages.

* Incorporated the request message behavioral clarifications from former Appendix C to Section 5. The definition of altCertTemplate was incorporated into Section 5.2.1, the clarification on POPOSigningKey was incorporated into Section 5.2.8, and the clarification on POPOPrivKey was incorporated into Section 5.2.8.1.

* Delete the mandatory algorithm profile in Appendix C.2 and refer instead to CMP Algorithms Section 7 [RFCCCCC].

1.3. Changes Made by This Document

This document obsoletes RFC 4210 [RFC4210]. It includes the changes specified by CMP Updates [RFCAAAA] Section 2 and Appendix C.2 as described in Section 1.2.

2. Requirements

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.

3. PKI Management Overview

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

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

3.1. PKI Management Model

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

3.1.1. Definitions of PKI Entities

The entities involved in PKI management include the end entity (i.e., the entity to whom the certificate is issued) and the certification authority (i.e., the entity that issues the certificate). A registration authority **MAY** also be involved in PKI management.

3.1.1.1. Subjects and End Entities

The term "subject" is used here to refer to the entity to whom the certificate is issued, typically named in the subject or subjectAltName field of a certificate. When we wish to distinguish the tools and/or software used by the subject (e.g., a local certificate management module), we will use the term "subject equipment". In general, the term "end entity" (EE), rather than "subject", is preferred in order to avoid confusion with the field name. It is important to note that the end entities here will include not only human users of applications, but also applications themselves (e.g., for IP security) or devices (e.g., routers or industrial control systems). This factor influences the protocols that the PKI management operations use; for example, application software is far more likely to know exactly which certificate extensions are required than are human users. PKI management entities are also end entities in the sense that they are sometimes named in the subject or subjectAltName field of a certificate or cross-certificate. Where appropriate, the term "end entity" will be used to refer to end entities who are not PKI management entities.

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

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

3.1.1.2. Certification Authority

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

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

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

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

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

3.1.1.3. Registration Authority

In addition to end-entities and CAs, many environments call for the existence of a Registration Authority (RA) separate from the Certification Authority. The functions that the registration authority may carry out will vary from case to case but **MAY** include personal authentication, token distribution, checking certificate requests and authentication of their origin, revocation reporting, name assignment, key generation, archival of key pairs, et cetera.

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

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

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

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

3.1.1.4. Key Generation Authority

A Key Generation Authority (KGA) is a PKI management entity generating key pairs on behalf of an end entity. Typically, such central key generation is performed by the CA itself. The KGA knows the private key that it generated for the end entity. The CA may delegate its authorization for generating key pairs on behalf of an end entity to another PKI management entity, such as an RA or a separate entity (see Section 4.5 for respective extended key usages).

Note: When doing central generation of key pairs, implementers should consider the implications of server-side retention on the overall security of the system; in some case retention is good, for example for escrow reasons, but in other cases the server should clear its copy after delivery to the end entity.

3.1.2. PKI Management Requirements

The protocols given here meet the following requirements on PKI management

1. PKI management must conform to the ISO/IEC 9594-8/ITU-T X.509 standards.

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

3. The use of confidentiality in PKI management protocols must be kept to a minimum in order to ease acceptance in environments where strong confidentiality might cause regulatory problems.

4. PKI management protocols must allow the use of different industry-standard cryptographic algorithms, see CMP Algorithms [RFCCCCC]. This means that any given CA, RA, or end entity may,

in principle, use whichever algorithms suit it for its own key pair(s).

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

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

7. PKI management protocols must support the production of Certificate Revocation Lists (CRLs) by allowing certified end entities to make requests for the revocation of certificates. This must be done in such a way that the denial-of-service attacks, which are possible, are not made simpler.

8. PKI management protocols must be usable over a variety of "transport" mechanisms, specifically including mail, HTTP, TCP/IP, CoAP, and off-line file-based.

9. Final authority for certification creation rests with the CA. No RA or end entity equipment can assume that any certificate issued by a CA will contain what was requested; a CA may alter certificate field values or may add, delete, or alter extensions according to its operating policy. In other words, all PKI entities (end-entities, RAs, and CAs) must be capable of handling responses to requests for certificates in which the actual certificate issued is different from that requested (for example, a CA may shorten the validity period requested). Note that policy may dictate that the CA must not publish or otherwise distribute the certificate until the requesting entity has reviewed and accepted the newly-created certificate or the indirect POP is completed (typically through use of the certConf message). In case of publication of the certificate or a precertificate in a Certificate Transparency log [RFC9162], the certificate must be revoked if it was not accepted or the indirect POP could not be completed.

10. A graceful, scheduled change-over from one non-compromised CA key pair to the next (CA key update) must be supported (note that if the CA key is compromised, re-initialization must be performed for all entities in the domain of that CA). An end entity whose PSE contains the new CA public key (following a CA key update) must also be able to verify certificates verifiable

using the old public key. End entities who directly trust the old CA key pair must also be able to verify certificates signed using the new CA private key (required for situations where the old CA public key is "hardwired" into the end entity's cryptographic equipment).

11. The functions of an RA may, in some implementations or environments, be carried out by the CA itself. The protocols must be designed so that end entities will use the same protocol regardless of whether the communication is with an RA or CA. Naturally, the end entity must use the correct RA or CA public key to protect the communication.

12. Where an end entity requests a certificate containing a given public key value, the end entity must be ready to demonstrate possession of the corresponding private key value. This may be accomplished in various ways, depending on the type of certification request. See Section 4.3 for details of the in-band methods defined for the PKIX-CMP (i.e., Certificate Management Protocol) messages.

3.1.3. PKI Management Operations

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

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

Figure 1: PKI Entities

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

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

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

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

    1. initial registration/certification: This is the process whereby an end entity first makes itself known to a CA or RA, prior to the CA issuing a certificate or certificates for that end entity. The end result of this process (when it is successful) is that a CA issues a certificate for an end entity's public key, and returns that certificate to the end entity and/or posts that certificate in a public repository. This process may, and typically will, involve multiple "steps", possibly including an initialization of the end entity's equipment. For example, the end entity's equipment must be securely initialized with the public key of a CA, to be used in validating certificate paths. Furthermore, an end entity typically needs to be initialized with its own key pair(s).

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

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

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

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

        1. Note 1. The above definition of "cross-certificate" aligns with the defined term "CA-certificate" in X.509. Note that this term is not to be confused with the X.500 "cACertificate" attribute type, which is unrelated.

2. Note 2. In many environments, the term "cross-certificate", unless further qualified, will be understood to be synonymous with "inter-domain cross-certificate" as defined above.

3. Note 3. Issuance of cross-certificates may be, but is not necessarily, mutual; that is, two CAs may issue cross-certificates for each other.

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

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

   1. certificate publication: Having gone to the trouble of producing a certificate, some means for publishing it is needed. The "means" defined in PKIX **MAY** involve the messages specified in Sections 5.3.13 to 5.3.16, or **MAY** involve other methods (LDAP, for example) as described in [RFC4510], [RFC4510] (the "Operational Protocols" documents of the PKIX series of specifications).

   2. CRL publication: As for certificate publication.

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

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

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

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

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

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

Later sections define a set of standard messages supporting the above operations. Transport protocols for conveying these exchanges in different environments (e.g., off-line: file-based, on-line: mail, HTTP [RFCDDDD], and CoAP [RFCEEEE]) are beyond the scope of this document and are specified separately.

4. Assumptions and Restrictions

4.1. End Entity Initialization

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

4.2. Initial Registration/Certification

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

However, we can classify the initial registration/certification schemes that are supported by this specification. Note that the word "initial", above, is crucial: we are dealing with the situation where the end entity in question has had no previous contact with the PKI. Where the end entity already possesses certified keys, then some simplifications/alternatives are possible.

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

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

4.2.1. Criteria Used

4.2.1.1. Initiation of Registration/Certification

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

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

4.2.1.2. End Entity Message Origin Authentication

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

In this specification, such authentication is achieved by two different means:

* symmetric: The PKI (CA/RA) issuing the end entity with a secret value (initial authentication key) and reference value (used to identify the secret value) via some out-of-band means. The initial authentication key can then be used to protect relevant PKI messages.

* asymmetric: Using a private key and certificate issued by another PKI trusted for initial authentication, e.g., an IDevID IEEE 802.1AR [IEEE.802.1AR-2018]. The trust establishment in this external PKI is out of scope of this document.

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

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

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

4.2.1.3. Location of Key Generation

In this specification, "key generation" is regarded as occurring wherever either the public or private component of a key pair first occurs in a PKIMessage. Note that this does not preclude a centralized key generation service by a KGA; the actual key pair **MAY** have been generated elsewhere and transported to the end entity, RA, or CA using a (proprietary or standardized) key generation request/response protocol (outside the scope of this specification).

Thus, there are three possibilities for the location of "key generation": the end entity, an RA, or a CA.

4.2.1.4. Confirmation of Successful Certification

Following the creation of an initial certificate for an end entity, additional assurance can be gained by having the end entity explicitly confirm successful receipt of the message containing (or indicating the creation of) the certificate. Naturally, this confirmation message must be protected (based on the initial symmetric or asymmetric authentication key or other means).

This gives two further possibilities: confirmed or not.

4.2.2. Mandatory Schemes

The criteria above allow for a large number of initial registration/certification schemes. This specification mandates that conforming CA equipment, RA equipment, and EE equipment **MUST** support the second scheme listed below (Section 4.2.2.2). Any entity **MAY** additionally support other schemes, if desired.

4.2.2.1. Centralized Scheme

In terms of the classification above, this scheme is, in some ways, the simplest possible, where:

* initiation occurs at the certifying CA;

* no on-line message authentication is required;

* "key generation" occurs at the certifying CA (see Section 4.2.1.3);

* no confirmation message is required.

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

4.2.2.2. Basic Authenticated Scheme

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

* initiation occurs at the end entity;

* message authentication is **REQUIRED**;

* "key generation" occurs at the end entity (see Section 4.2.1.3);

* a confirmation message is **REQUIRED**.

Note: An Initial Authentication Key (IAK) can be either a symmetric key or an asymmetric private key with a certificate issued by another PKI trusted for this purpose. The establishment of such trust is out of scope of this document.

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

```
 End entity                                         RA/CA
==========                                      =============
     out-of-band distribution of Initial Authentication
     Key (IAK) and reference value (RA/CA -> EE)
Key generation
Creation of certification request
Protect request with IAK
              -->>-- certification request -->>--
                                               verify request
                                               process request
                                               create response
              --<<-- certification response --<<--
handle response
create confirmation
              -->>-- cert conf message      -->>--
                                               verify confirmation
                                               create response
              --<<-- conf ack (optional)    --<<--
handle response
```

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

4.3. Proof-of-Possession (POP) of Private Key

< ToDo: To be aligned with Section 5.2.8 if needed. >

In order to prevent certain attacks and to allow a CA/RA to properly check the validity of the binding between an end entity and a key pair, the PKI management operations specified here make it possible for an end entity to prove that it has possession of (i.e., is able to use) the private key corresponding to the public key for which a certificate is requested. A given CA/RA is free to choose how to enforce POP (e.g., out-of-band procedural means versus PKIX-CMP in-band messages) in its certification exchanges (i.e., this may be a policy issue). However, it is **REQUIRED** that CAs/RAs **MUST** enforce POP by some means because there are currently many non-PKIX operational protocols in use (various electronic mail protocols are one example) that do not explicitly check the binding between the end entity and the private key. Until operational protocols that do verify the binding (for signature, encryption, and key agreement key pairs) exist, and are ubiquitous, this binding can only be assumed to have been verified by the CA/RA. Therefore, if the binding is not verified by the CA/RA, certificates in the Internet Public-Key Infrastructure end up being somewhat less meaningful.

POP is accomplished in different ways depending upon the type of key for which a certificate is requested. If a key can be used for multiple purposes (e.g., an RSA key) then any appropriate method **MAY** be used (e.g., a key that may be used for signing, as well as other purposes, **SHOULD NOT** be sent to the CA/RA in order to prove possession).

This specification explicitly allows for cases where an end entity supplies the relevant proof to an RA and the RA subsequently attests to the CA that the required proof has been received (and validated!). For example, an end entity wishing to have a signing key certified could send the appropriate signature to the RA, which then simply notifies the relevant CA that the end entity has supplied the required proof. Of course, such a situation may be disallowed by some policies (e.g., CAs may be the only entities permitted to verify POP during certification).

4.3.1. Signature Keys

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

4.3.2. Encryption Keys

For encryption keys, the end entity can provide the private key to the CA/RA, or can be required to decrypt a value in order to prove possession of the private key (see Section 5.2.8). Decrypting a value can be achieved either directly or indirectly.

The direct method is for the RA/CA to issue a random challenge to which an immediate response by the EE is required.

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

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

4.3.3. Key Agreement Keys

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

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

4.3.4. Key Encapsulation Mechanism Keys

For key encapsulation mechanism keys, the end entity can be required to decrypt a value in order to prove possession of the private key (see Section 5.2.8). Decrypting a value can be achieved either directly or indirectly.

Note: A definition of Key Encapsulation Mechanisms can be found in [I-D.ietf-lamps-cms-kemri], Section 1.

The direct method is for the RA/CA to issue a random challenge to which an immediate response by the EE is required.

The indirect method is to issue a certificate that is encrypted for the end entity using a shared secret key derived from a key encapsulated using the public key (and have the end entity demonstrate its ability to use its private key for decapsulation of the KEM ciphertext, derive the shared secret key, decrypt this certificate, and provide a hash of the certificate in the confirmation message). This allows a CA to issue a certificate in a form that can only be used by the intended end entity.

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

4.4. Root CA Key Update

This discussion only applies to CAs that are directly trusted by some end entities. Self-signed CAs **SHALL** be considered as directly trusted CAs. Recognizing whether a non-self-signed CA is supposed to be directly trusted for some end entities is a matter of CA policy and is thus beyond the scope of this document.

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

When a CA changes its key pair, those entities who have acquired the old CA public key via "out-of-band" means are most affected. It is these end entities who will need access to the new CA public key protected with the old CA private key. However, they will only require this for a limited period (until they have acquired the new CA public key via the "out-of-band" mechanism). This will typically be easily achieved when these end entities' certificates expire.

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

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

Note 2:. While the scheme could be generalized to cover cases where the CA updates its key pair more than once during the validity period of one of its end entities' certificates, this generalization seems of dubious value. Not having this generalization simply means that the validity periods of certificates issued with the old CA key pair cannot exceed the end of the OldWithNew validity period.

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

Note 4: In practice, a new root CA may have a slightly different subject DN, e.g., indicating a generation identifier like the year of issuance or a version number, for instance in an OU element. How to bridge trust to the new root CA certificate in a CA DN change or a cross-certificate scenario is out of scope for this document.

4.4.1. CA Operator Actions

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

1. Generate a new key pair;

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

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

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

5. Publish these new certificates via the repository and/or other means (perhaps using a CAKeyUpdAnn message or RootCaKeyUpdateContent);

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

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

The "old with new" certificate must have a validity period with the same notBefore and notAfter time as the "old with old" certificate.

The "new with old" certificate must have a validity period with the same notBefore time as the "new with new" certificate and a notAfter time by which all end entities of this CA will securely possess the new CA public key (at the latest, at the notAfter time of the "old with old" certificate).

The "new with new" certificate must have a validity period with a notBefore time that is before the notAfter time of the "old with old" certificate and a notAfter time that is after the notBefore time of the next update of this certificate.

Note: Further operational considerations on transition from one root CA self-signed certificate to the next is available in RFC 8649 Section 5 [RFC8649].

4.4.2. Verifying Certificates

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

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

Note: Instead of using a repository, the end entity can use the root CA update general message to request the respective certificates from a PKI management entity, see Section 5.3.19.15, and follow the required validation steps.

4.4.2.1. Verification in Cases 1, 4, 5, and 8

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

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

4.4.2.2. Verification in Case 2

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

1. Look up the caCertificate attribute in the repository and pick the OldWithNew certificate (determined based on validity periods; note that the subject and issuer fields must match);

2. Verify that this is correct using the new CA key (which the verifier has locally);

3. If correct, check the signer's certificate using the old CA key.

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

4.4.2.3. Verification in Case 3

In case 3, the verifier must get access to the new public key of the CA. In case a repository is used, the verifier does the following:

1. Look up the cACertificate attribute in the repository and pick the NewWithOld certificate (determined based on validity periods; note that the subject and issuer fields must match);

2. Verify that this is correct using the old CA key (which the verifier has stored locally);

3. If correct, check the signer's certificate using the new CA key.

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

Note: Alternatively, the verifier can use the root CA update general message to request the respective certificates from a PKI management entity, see Section 5.3.19.15, and follow the required validation steps.

4.4.2.4. Failure of Verification in Case 6

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

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

4.4.2.5. Failure of Verification in Case 7

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

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

4.4.3. Revocation - Change of CA Key

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

The analysis of the alternatives is the same as for certificate verification.

4.5. Extended Key Usage

The Extended Key Usage (EKU) extension indicates the purposes for which the certified key pair may be used. It therefore restricts the use of a certificate to specific applications.

A CA may want to delegate parts of its duties to other PKI management entities. This section provides a mechanism to both prove this delegation and enable automated means for checking the authorization of this delegation. Such delegation may also be expressed by other means, e.g., explicit configuration.

To offer automatic validation for the delegation of a role by a CA to another entity, the certificates used for CMP message protection or signed data for central key generation **MUST** be issued by the delegating CA and **MUST** contain the respective EKUs. This proves the authorization of this entity by the delegating CA to act in the given role as described below.

The OIDs to be used for these EKUs are:

```
id-kp-cmcCA OBJECT IDENTIFIER ::= {
   iso(1) identified-organization(3) dod(6) internet(1)
   security(5) mechanisms(5) pkix(7) kp(3) 27 }

id-kp-cmcRA OBJECT IDENTIFIER ::= {
   iso(1) identified-organization(3) dod(6) internet(1)
   security(5) mechanisms(5) pkix(7) kp(3) 28 }

id-kp-cmKGA OBJECT IDENTIFIER ::= {
   iso(1) identified-organization(3) dod(6) internet(1)
   security(5) mechanisms(5) pkix(7) kp(3) 32 }
```

Note: RFC 6402 section 2.10 [RFC6402] specifies OIDs for a CMC CA and a CMC RA. As the functionality of a CA and RA is not specific to using CMC or CMP as the certificate management protocol, these EKUs are re-used by CMP.

The meaning of the id-kp-cmKGA EKU is as follows:

CMP KGA: CMP Key Generation Authorities are CAs or are identified by the id-kp-cmKGA extended key usage. The CMP KGA knows the private key it generated on behalf of the end entity. This is a very sensitive service and needs specific authorization, which by default is with the CA certificate itself. The CA may delegate its authorization by placing the id-kp-cmKGA extended key usage in the certificate used to authenticate the origin of the generated private key. The authorization may also be determined through local configuration of the end entity.

5. Data Structures

This section contains descriptions of the data structures required for PKI management messages. Section 6 describes constraints on their values and the sequence of events for each of the various PKI management operations.

5.1. Overall PKI Message

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

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

PKIMessages ::= SEQUENCE SIZE (1..MAX) OF PKIMessage
```

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

The PKIBody contains message-specific information.

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

The extraCerts field can contain certificates that may be useful to the recipient. For example, this can be used by a CA or RA to present an end entity with certificates that it needs to verify its own new certificate (if, for example, the CA that issued the end entity's certificate is not a root CA for the end entity). Note that this field does not necessarily contain a certification path; the recipient may have to sort, select from, or otherwise process the extra certificates in order to use them.

5.1.1. PKI Message Header

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

The following data structure is used to contain this information:

```
PKIHeader ::= SEQUENCE {
   pvno                INTEGER     { cmp1999(1), cmp2000(2),
                                     cmp2021(3) },
   sender              GeneralName,
   recipient           GeneralName,
   messageTime     [0] GeneralizedTime         OPTIONAL,
   protectionAlg   [1] AlgorithmIdentifier{ALGORITHM, {...}}
                       OPTIONAL,
   senderKID       [2] KeyIdentifier           OPTIONAL,
   recipKID        [3] KeyIdentifier           OPTIONAL,
   transactionID   [4] OCTET STRING            OPTIONAL,
   senderNonce     [5] OCTET STRING            OPTIONAL,
   recipNonce      [6] OCTET STRING            OPTIONAL,
   freeText        [7] PKIFreeText             OPTIONAL,
   generalInfo     [8] SEQUENCE SIZE (1..MAX) OF
                       InfoTypeAndValue     OPTIONAL
}

PKIFreeText ::= SEQUENCE SIZE (1..MAX) OF UTF8String
```

The usage of pvno values is described in Section 7.

The sender field contains the name of the sender of the PKIMessage. This name (in conjunction with senderKID, if supplied) should be sufficient to indicate the key to use to verify the protection on the message. If nothing about the sender is known to the sending entity (e.g., in the init. req. message, where the end entity may not know its own Distinguished Name (DN), e-mail name, IP address, etc.), then the "sender" field **MUST** contain a "NULL" value; that is, the SEQUENCE OF relative distinguished names is of zero length. In such a case, the senderKID field **MUST** hold an identifier (i.e., a reference number) that indicates to the receiver the appropriate shared secret information to use to verify the message.

The recipient field contains the name of the recipient of the PKIMessage. This name (in conjunction with recipKID, if supplied) should be usable to verify the protection on the message where the recipient's KEM key is used.

The protectionAlg field specifies the algorithm used to protect the message. If no protection bits are supplied (note that PKIProtection is **OPTIONAL**) then this field **MUST** be omitted; if protection bits are supplied, then this field **MUST** be supplied.

senderKID and recipKID are usable to indicate which keys have been used to protect the message (recipKID will normally only be required where protection of the message also uses the recipient's KEM key). These fields **MUST** be used if required to uniquely identify a key (e.g., if more than one key is associated with a given sender name). The senderKID **SHOULD** be used in any case.

Note: The recommendation of using senderKID was changed since [RFC4210], where it was recommended to be omitted if not needed to identify the protection key.

The transactionID field within the message header is to be used to allow the recipient of a message to correlate this with an ongoing transaction. This is needed for all transactions that consist of more than just a single request/response pair. For transactions that consist of a single request/response pair, the rules are as follows. A client **MAY** populate the transactionID field of the request. If a server receives such a request that has the transactionID field set, then it **MUST** set the transactionID field of the response to the same value. If a server receives such request with a missing transactionID field, then it **MAY** set transactionID field of the response.

For transactions that consist of more than just a single request/response pair, the rules are as follows. Clients **SHOULD** generate a transactionID for the first request. If a server receives such a request that has the transactionID field set, then it **MUST** set the transactionID field of the response to the same value. If a server receives such request with a missing transactionID field, then it **MUST** populate the transactionID field of the response with a server-generated ID. Subsequent requests and responses **MUST** all set the transactionID field to the thus established value. In all cases where a transactionID is being used, a given client **MUST NOT** have more than one transaction with the same transactionID in progress at any time (to a given server). Servers are free to require uniqueness of the transactionID or not, as long as they are able to correctly associate messages with the corresponding transaction. Typically, this means that a server will require the {client, transactionID} tuple to be unique, or even the transactionID alone to be unique, if it cannot distinguish clients based on transport-level information. A server receiving the first message of a transaction (which requires more than a single request/response pair) that contains a transactionID that does not allow it to meet the above constraints (typically because the transactionID is already in use) **MUST** send back an ErrorMsgContent with a PKIFailureInfo of transactionIdInUse. It is **RECOMMENDED** that the clients fill the transactionID field with 128 bits of (pseudo-) random data for the start of a transaction to reduce the probability of having the transactionID in use at the server.

The senderNonce and recipNonce fields protect the PKIMessage against replay attacks. The senderNonce will typically be 128 bits of (pseudo-) random data generated by the sender, whereas the recipNonce is copied from the senderNonce of the previous message in the transaction.

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

The freeText field may be used to send a human-readable message to the recipient (in any number of languages). The first language used in this sequence indicates the desired language for replies.

The generalInfo field may be used to send machine-processable additional data to the recipient. The following generalInfo extensions are defined and **MAY** be supported.

5.1.1.1. ImplicitConfirm

This is used by the EE to inform the CA that it does not wish to send a certificate confirmation for issued certificates.

```
id-it-implicitConfirm OBJECT IDENTIFIER ::= {id-it 13}
ImplicitConfirmValue ::= NULL
```

If the CA grants the request to the EE, it **MUST** put the same extension in the PKIHeader of the response. If the EE does not find the extension in the response, it **MUST** send the certificate confirmation.

5.1.1.2. ConfirmWaitTime

This is used by the CA to inform the EE how long it intends to wait for the certificate confirmation before revoking the certificate and deleting the transaction.

```
id-it-confirmWaitTime OBJECT IDENTIFIER ::= {id-it 14}
ConfirmWaitTimeValue ::= GeneralizedTime
```

5.1.1.3. OrigPKIMessage

An RA **MAY** include the original PKIMessage from the EE in this generalInfo field of the PKIHeader of a PKIMessage. This is used by the RA to inform the CA of the original PKIMessage that it received from the EE and modified in some way (e.g., added or modified particular field values or added new extensions) before forwarding the new PKIMessage. If the changes made by the RA to the original PKIMessage break the POP of a certificate request, the RA **MUST** set the popo field to RAVerified, see Section 5.2.8.4. This accommodates, for example, cases in which the CA wishes to check POP or other information on the original EE message.

Although the infoValue is PKIMessages, it **MUST** contain exactly one PKIMessage.

```
id-it-origPKIMessage OBJECT IDENTIFIER ::= {id-it 15}
OrigPKIMessageValue ::= PKIMessages
```

5.1.1.4. CertProfile

This is used by the EE to indicate specific certificate profiles, e.g., when requesting a new certificate or a certificate request template, see Section 5.3.19.16.

```
id-it-certProfile OBJECT IDENTIFIER ::= {id-it 21}
CertProfileValue ::= SEQUENCE SIZE (1..MAX) OF UTF8String
```

When used in an ir/cr/kur/genm, the value **MUST NOT** contain more elements than the number of CertReqMsg or InfoTypeAndValue elements and the certificate profile names refer to the elements in the given order.

When used in a p10cr, the value **MUST NOT** contain multiple certificate profile names.

5.1.2. PKI Message Body

```
PKIBody ::= CHOICE {
   ir       [0]  CertReqMessages,        --Initialization Req
   ip       [1]  CertRepMessage,         --Initialization Resp
   cr       [2]  CertReqMessages,        --Certification Req
   cp       [3]  CertRepMessage,         --Certification Resp
   p10cr    [4]  CertificationRequest,   --PKCS #10 Cert.  Req.
   popdecc  [5]  POPODecKeyChallContent --pop Challenge
   popdecr  [6]  POPODecKeyRespContent, --pop Response
   kur      [7]  CertReqMessages,        --Key Update Request
   kup      [8]  CertRepMessage,         --Key Update Response
   krr      [9]  CertReqMessages,        --Key Recovery Req
   krp      [10] KeyRecRepContent,       --Key Recovery Resp
   rr       [11] RevReqContent,          --Revocation Request
   rp       [12] RevRepContent,          --Revocation Response
   ccr      [13] CertReqMessages,        --Cross-Cert.  Request
   ccp      [14] CertRepMessage,         --Cross-Cert.  Resp
   ckuann   [15] CAKeyUpdAnnContent,     --CA Key Update Ann.
   cann     [16] CertAnnContent,         --Certificate Ann.
   rann     [17] RevAnnContent,          --Revocation Ann.
   crlann   [18] CRLAnnContent,          --CRL Announcement
   pkiconf  [19] PKIConfirmContent,      --Confirmation
   nested   [20] NestedMessageContent,   --Nested Message
   genm     [21] GenMsgContent,          --General Message
   genp     [22] GenRepContent,          --General Response
   error    [23] ErrorMsgContent,        --Error Message
   certConf [24] CertConfirmContent,     --Certificate confirm
   pollReq  [25] PollReqContent,         --Polling request
   pollRep  [26] PollRepContent          --Polling response
}
```

The specific types are described in Section 5.3 below.

5.1.3. PKI Message Protection

Some PKI messages will be protected for integrity.

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

When protection is applied, the following structure is used:

```
PKIProtection ::= BIT STRING
```

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

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

There **MAY** be cases in which the PKIProtection BIT STRING is deliberately not used to protect a message (i.e., this **OPTIONAL** field is omitted) because other protection, external to PKIX, will be applied instead. Such a choice is explicitly allowed in this specification. Examples of such external protection include CMS [RFC5652] and Security Multiparts [RFC1847] encapsulation of the PKIMessage (or simply the PKIBody (omitting the CHOICE tag), if the relevant PKIHeader information is securely carried in the external mechanism). It is noted, however, that many such external mechanisms require that the end entity already possesses a public-key certificate, and/or a unique Distinguished Name, and/or other such infrastructure-related information. Thus, they may not be appropriate for initial registration, key-recovery, or any other process with "boot-strapping" characteristics. For those cases it may be necessary that the PKIProtection parameter be used. In the future, if/when external mechanisms are modified to accommodate boot-strapping scenarios, the use of PKIProtection may become rare or non-existent.

Depending on the circumstances, the PKIProtection bits may contain a Message Authentication Code (MAC) or signature. Only the following cases can occur:

5.1.3.1. Pre-Shared Secret Information

In this case, the sender and recipient share secret information with sufficient entropy (established via out-of-band means). PKIProtection will contain a MAC value and the protectionAlg **MAY** be one of the options described in CMP Algorithms Section 6.1 [RFCCCCC].

5.1.3.2. Key Agreement

Where the sender and receiver possess finite-field or elliptic-curve-based Diffie-Hellman certificates with compatible DH parameters, in order to protect the message the end entity must generate a symmetric key based on its private DH key value and the DH public key of the recipient of the PKI message. PKIProtection will contain a MAC value keyed with this derived symmetric key and the protectionAlg will be the following:

```
id-DHBasedMac OBJECT IDENTIFIER ::= {1 2 840 113533 7 66 30}

DHBMParameter ::= SEQUENCE {
   owf                 AlgorithmIdentifier,
   -- AlgId for a One-Way Function
   mac                 AlgorithmIdentifier
   -- the MAC AlgId
}
```

In the above protectionAlg, OWF is applied to the result of the Diffie-Hellman computation. The OWF output (called "BASEKEY" for ease of reference, with a size of "H") is what is used to form the symmetric key. If the MAC algorithm requires a K-bit key and K <= H, then the most significant K bits of BASEKEY are used. If K > H, then all of BASEKEY is used for the most significant H bits of the key, OWF("1" || BASEKEY) is used for the next most significant H bits of the key, OWF("2" || BASEKEY) is used for the next most significant H bits of the key, and so on, until all K bits have been derived. [Here "N" is the ASCII byte encoding the number N and "||" represents concatenation.]

Note: Hash algorithms that can be used as one-way functions are listed in CMP Algortihms [RFCCCCC] Section 2.

Note: As alternative to this mechanism, the mechanism described in Section 5.1.3.4 can be applies.

5.1.3.3. Signature

In this case, the sender possesses a signature key pair and simply signs the PKI message. PKIProtection will contain the signature value and the protectionAlg will be an AlgorithmIdentifier for a digital signature **MAY** be one of the options described in CMP Algorithms Section 3 [RFCCCCC].

5.1.3.4. Key Encapsulation

< ToDo: Version -05 proposed an approach using HPKE to establish an authenticated shared symmetric key. This version offers an alternative using plain KEM and KDF functions as shown in draft-ietf-lamps-cms-kemri [I-D.ietf-lamps-cms-kemri]. >

< ToDo: Version -05 as well as this version utilizes keys from both CMP client and server in applying Encapsulate/Decapsulate twice. Deriving a joint key from the two shared secrets result in a mutually authenticated shared symmetric key used by both parties, like when using an out-of-band established shared secret information. In contrast to this approach, each side could directly use the shared secret resulting from the Encapsulate/Decapsulate for deriving an individual shared symmetric key. Doing so the MAC-based

protection generated on both sides would use a different key. This would facilitate that only one side used a KEM key pair and the other side uses a signature key pair. This would offer some additional flexibility, but on at the price of a more complex key establishment. >

In this case, an initial CMP message exchange using general messages is required to contribute to establishing a shared symmetric key (ssk) as follows. Both PKI entities require a KEM certificate of the other side. Each sender uses the KEM Encapsulate(pk) -> (ss, ct) function with the recipient's public key (pk) to produce the KEM shared secret (ss) and an encapsulation of that shared secret, the KEM ciphertext (ct) and provide the ciphertext to the recipient using the id-it-KemCiphertext as defined below. The respective recipient uses a Decapsulate(sk, ct)->(ss) function with the private key (sk) and the ciphertext (ct) to recover the the KEM shared secret (ss) from the encapsulated representation received from the sender. Each party uses a KDF(ikm, len, ukm)->(ssk) function with input key material (ikm), the desired length of the output keying material (len), and user key material (ukm) to derive a shares symmetric key (ssk). The concatenation of both shared secrets (ss1) and (ss2) are the first argument (ikm) for the KDF. In addition a static string and some random data from the PKI message header also contribute to the third argument (ukm) for the KDF. Doing so, a shared symmetric key (ssk) authenticated by both PKI entities is established and can be used for MAC-based message protection.

Note: This approach uses the definition of Key Encapsulation Mechanisms in [I-D.ietf-lamps-cms-kemri], Section 1.

The InfoTypeAndValue transferring the KEM ciphertext is of type id-it-KemCiphertext, which is defined in this document as:

```
id-it-KemCiphertext OBJECT IDENTIFIER ::= { id-it TBD1 }
```

Note: This InfoTypeAndValue can be carried in genm/genp message body or in the generalInfo field of PKIHeader.

When id-it-KemCiphertext is used, the value is either absent or of type KemCiphertext. The syntax for KemCiphertext is as follows:

```
KemCiphertext ::= SEQUENCE {
  kem              AlgorithmIdentifier,
  -- AlgId of the Key Encapsulation Mechanism
  kdf              AlgorithmIdentifier,
  -- AlgId of the Key Derivation Function
  len              INTEGER,
  -- Defines the length of the keying material output of the KDF
  -- SHOULD be the maximum key length of the MAC function
  -- MUST NOT be larger that 255*Nh of the KDF where Nh is the
  --   output size of the KDF
  ct               OCTET STRING
  -- Ciphertext output from the Encapsulate function
}
```

< ToDo: Update the ASN.1 module! >

When CMP messages protection using the shared symmetric key (ssk) derived from the two KEM shared secrets the MAC algorithm is placed in the protectionAlg header field and the MAC value is placed in the PKIProtection field. The MAC algorithm **MAY** be one of the options described in CMP Algorithms Section 6.2 [RFCCCCC].

The message flow using KEM keys for message protection is as follows. A regular CMP transaction is preceded by a genm/genp message exchange with no protection that is needed to establish a shared symmetric key using KEM public keys from the sender's and recipient's KEM certificate. This shared symmetric key (ssk) is then used for MAC-based protection of the remaining request and response messages of the transaction.

Message Flow:

```
Step# PKI entity                          PKI management entity
      (client role)                       (server role)
  1   format genm without
        protection
                         ->   genm    ->
  2                                       validate certificate of
                                            PKI entity
                                          perform KEM Encapsulate
                                          format genp without
                                            protection
                         <-   genp    <-
  3   validate
        certificate of PKI
        management entity
      perform KEM Decapsulate
      perform KEM Encapsulate
      perform key derivation
      format request with
        MAC-based protection
                         ->  request  ->
  4                                       perform KEM Decapsulate
                                          perform key derivation
                                          verify MAC-based
                                            protection
----------------  client authenticated by the server  ---------------
                                          format response with
                                            MAC-based protection
                         <-  response <-
  5   verify MAC-based protection
----------------  server authenticated by the client  ---------------
        Further messages can be exchanged with MAC-based
      protection using the established shared symmetric key
```

Figure 2: Message flow establishing a shared symmetric key for MAC-based protection

1. The PKI entity in Figure 2, which is the client, formats a genm message of type id-it-KemCiphertext without a value in its body. The message has no protection, and the extraCerts field contains the KEM certificate of the client.

2. The server validates the client KEM certificate.

   It generates a shared secret ss1 and the associated ciphertext ct1 using the KEM Encapsulate function with the client's public key pkC:

```
Encapsulate(pkC) -> (ct1, ss1)
```

The PKI management entity in Figure 2, which is the server, formats a genp message of type id-it-KemCiphertext with a value of type KemCiphertext containing OIDs of the used KEM and KDF algorithms, the desired output length len of the KDF, and the KEM ciphertext ct1. The message has no protection, and the extraCerts field contains the KEM certificate of the server.

3. The PKI entity validates the server KEM certificate.

   It decapsulates the shared secret ss1 from the ciphertext ct1 using the KEM Decapsulate function and its private key skC:

   ```
   Decapsulate(ct1, skC) -> (ss1)
   ```

   Note: If the decapsulation operation outputs an error, any PKIFailureInfo **SHALL** contain the value badMessageCheck and the PKI management operation **SHALL** be terminated.

   It generates a shared secret ss2 and the associated ciphertext ct2 using the KEM Encapsulate function with the server's public key pkS:

   ```
   Encapsulate(pkS) = (ct2, ss2)
   ```

   It concatenates the shared secret ss1, with the shared secret ss2 to the input key material ikm.

   ```
   ikm = concat(ss1, ss2)
   ```

   It concatenates the static text "CMP-KEM", the transactionID, the senderNonce genp_senderNonce, and the recipNonce genp_recipNonce from the PKIHeader of the genp message to the user key material ukm.

   Note: The function concat(x0, ..., xN) concatenates the byte strings as specified in RFC 9180 Section 3 [RFC9180].

   ```
   ukm = concat("CMP-KEM", transactionID, genp_senderNonce,
                genp_recipNonce)
   ```

   It derives the shared symmetric key ssk from the input key material ikm using the desired output length len and the user key material ukm.

   ```
   KDF(ikm, len, ukm)->(ssk)
   ```

   Note: The shared symmetric key ssk is used for MAC-based protection of all subsequent messages of this PKI management operation. This construction for combining two KEM shared secrets using a KDF is at least as strong as the KEM combiner presented in [I-D.ounsworth-cfrg-kem-combiners], also see Section 8.8 for further discussion.

< ToDo: @Mike, please check the combining of the two shared secrets and if the above note is still correct and needed. >

The request message is of any type. The generalInfo field in the message header contains an element of type id-it-KemCiphertext and the value is of type KemCiphertext containing OIDs of the used KEM and KDF algorithm, the desired output length len of the KDF, and the KEM ciphertext ct2. The message has a MAC-based protection using the shared symmetric key ssk.

4. The PKI management entity decapsulates the shared secret ss2 from the ciphertext ct2 using the Decapsulate function and its private key skS:

```
Decapsulate(ct2, skS) = ss2
```

Note: If the decapsulation operation outputs an error, The PKI management entity **SHALL** return a PKIFailureInfo containing the value badMessageCheck and the PKI management operation **SHALL** be terminated.

It concatenates the shared secret ss1, with the shared secret ss2 to the input key material ikm.

```
ikm = concat(ss1, ss2)
```

It concatenates the static text "CMP-KEM", the transactionID, the senderNonce genp_senderNonce, and the recipNonce genp_recipNonce from the PKIHeader of the genp message to the user key material ukm.

```
ukm = concat("CMP-KEM", transactionID, genp_senderNonce,
             genp_recipNonce)
```

It derives the shared symmetric key ssk from the input key material ikm using the desired output length len and the user key material ukm.

```
KDF(ikm, len, ukm)->(ssk)
```

It verifies the MAC-based protection and thus authenticates the PKI entity as the sender of the request message.

The response message also has a MAC-based protection using the shared symmetric key ssk.

5. The PKI entity verifies the MAC-based protection and thus authenticates the PKI management entity as the sender of the response message.

All potential further message of this PKI management operation make use of the shared symmetric key ssk for MAC-based protection.

5.1.3.5. Multiple Protection

When receiving a protected PKI message, a PKI management entity such as an RA **MAY** forward that message adding its own protection (which is a MAC or a signature, depending on the information and certificates shared between the RA and the CA). Additionally, multiple PKI messages **MAY** be aggregated. There are several use cases for such messages.

* The RA confirms having validated and authorized a message and forwards the original message unchanged.

* A PKI management entity collects several messages that are to be forwarded in the same direction and forwards them in a batch. Request messages can be transferred as batch upstream (towards the CA); response or announce messages can be transferred as batch downstream (towards an RA, but not to the EE). This can for instance be used when bridging an off-line connection between two PKI management entities.

These use cases are accomplished by nesting the messages within a new PKI message. The structure used is as follows:

```
NestedMessageContent ::= PKIMessages
```

5.2. Common Data Structures

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

5.2.1. Requested Certificate Contents

Various PKI management messages require that the originator of the message indicate some of the fields that are required to be present in a certificate. The CertTemplate structure allows an end entity or RA to specify as much as it wishes about the certificate it requires. CertTemplate is identical to a Certificate, but with all fields optional.

Note: Even if the originator completely specifies the contents of a certificate it requires, a CA is free to modify fields within the certificate actually issued. If the modified certificate is unacceptable to the requester, the requester **MUST** send back a certConf message that either does not include this certificate (via a CertHash), or does include this certificate (via a CertHash) along with a status of "rejected". See Section 5.3.18 for the definition and use of CertHash and the certConf message.

Note: Before requesting a new certificate, an end entity can request a certTemplate structure as a kind of certificate request blueprint, in order to learn which data the CA expects to be present in the certificate request, see Section 5.3.19.16.

See CRMF [RFC4211] for CertTemplate syntax.

If certTemplate is an empty SEQUENCE (i.e., all fields omitted), then the controls field in the CertRequest structure **MAY** contain the id-regCtrl-altCertTemplate control, specifying a template for a certificate other than an X.509v3 public-key certificate. Conversely, if certTemplate is not empty (i.e., at least one field is present), then controls **MUST NOT** contain id-regCtrl-altCertTemplate. The new control is defined as follows:

```
id-regCtrl-altCertTemplate OBJECT IDENTIFIER ::= { iso(1)
   identified-organization(3) dod(6) internet(1) security(5)
   mechanisms(5) pkix(7) pkip(5) regCtrl(1) 7}

AltCertTemplate ::= AttributeTypeAndValue
```

5.2.2. Encrypted Values

Where encrypted data (in this specification, private keys, certificates, or revocation passphrase) are sent in PKI messages, the EncryptedKey data structure is used.

```
EncryptedKey ::= CHOICE {
   encryptedValue       EncryptedValue, -- deprecated
   envelopedData    [0] EnvelopedData }
```

See CRMF [RFC4211] for EncryptedKey and EncryptedValue syntax and CMS [RFC5652] for EnvelopedData syntax. Using the EncryptedKey data structure offers the choice to either use EncryptedValue (for backward compatibility only) or EnvelopedData. The use of the EncryptedValue structure has been deprecated in favor of the EnvelopedData structure. Therefore, it is **RECOMMENDED** to use EnvelopedData.

Note: The EncryptedKey structure defined in CRMF [RFC4211] is used here, which makes the update backward compatible. Using the new syntax with the untagged default choice EncryptedValue is bits-on-the-wire compatible with the old syntax.

To indicate support for EnvelopedData the pvno cmp2021 has been introduced. Details on the usage of pvno values is described in Section 7.

The EncryptedKey data structure is used in CMP to transport a private key, certificate, or revocation passphrase in encrypted form.

EnvelopedData is used as follows:

* It contains only one RecipientInfo structure because the content is encrypted only for one recipient.

* It may contain a private key in the AsymmetricKeyPackage structure as defined in RFC 5958 [RFC5958] wrapped in a SignedData structure as specified in CMS section 5 [RFC5652] and [RFC8933] signed by the Key Generation Authority.

* It may contain a certificate or revocation passphrase directly in the encryptedContent field.

The content of the EnvelopedData structure, as specified in CMS section 6 [RFC5652], **MUST** be encrypted using a newly generated symmetric content-encryption key. This content-encryption key **MUST** be securely provided to the recipient using one of three key management techniques.

The choice of the key management technique to be used by the sender depends on the credential available at the recipient:

* Recipient's certificate with an algorithm identifier and a public key that supports key transport and where any given key usage extension allows keyEncipherment: The content-encryption key will be protected using the key transport key management technique, as specified in CMS Section 6.2.1 [RFC5652].

* Recipient's certificate with an algorithm identifier and a public key that supports key agreement and where any given key usage extension allows keyAgreement: The content-encryption key will be protected using the key agreement key management technique, as specified in CMS Section 6.2.2 [RFC5652]. This is the preferred technique.

* A password or shared secret: The content-encryption key will be protected using the password-based key management technique, as specified in CMS Section 6.2.4 [RFC5652].

* Recipient's certificate with an algorithm identifier and a public key that supports key encapsulation mechanism and where any given key usage extension allows keyEncipherment: The content-

encryption key will be protected using the additional key management technique for KEM keys, as specified in [I-D.ietf-lamps-cms-kemri].

Note: There are cases where the algorithm identifier, the type of the public key, and the key usage extension will not be sufficient to decide on the key management technique to use, e.g., when rsaEncryption is the algorithm identifier. In such cases it is a matter of local policy to decide.

5.2.3. Status codes and Failure Information for PKI Messages

All response messages will include some status information. The following values are defined.

```
PKIStatus ::= INTEGER {
   accepted               (0),
   grantedWithMods        (1),
   rejection              (2),
   waiting                (3),
   revocationWarning      (4),
   revocationNotification (5),
   keyUpdateWarning       (6)
}
```

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

```
PKIFailureInfo ::= BIT STRING {
   badAlg               (0),
   badMessageCheck      (1),
   badRequest           (2),
   badTime              (3),
   badCertId            (4),
   badDataFormat        (5),
   wrongAuthority       (6),
   incorrectData        (7),
   missingTimeStamp     (8),
   badPOP               (9),
   certRevoked          (10),
   certConfirmed        (11),
   wrongIntegrity       (12),
   badRecipientNonce    (13),
   timeNotAvailable     (14),
   unacceptedPolicy     (15),
   unacceptedExtension  (16),
   addInfoNotAvailable  (17),
   badSenderNonce       (18),
   badCertTemplate      (19),
   signerNotTrusted     (20),
   transactionIdInUse   (21),
   unsupportedVersion   (22),
   notAuthorized        (23),
   systemUnavail        (24),
   systemFailure        (25),
   duplicateCertReq     (26)
}

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

5.2.4. Certificate Identification

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

See [RFC4211] for CertId syntax.

5.2.5. Out-of-band root CA Public Key

Each root CA must be able to publish its current public key via some "out-of-band" means. While such mechanisms are beyond the scope of this document, we define data structures that can support such mechanisms.

There are generally two methods available: either the CA directly publishes its self-signed certificate, or this information is available via the Directory (or equivalent) and the CA publishes a hash of this value to allow verification of its integrity before use.

```
OOBCert ::= Certificate
```

The fields within this certificate are restricted as follows:

* The certificate **MUST** be self-signed (i.e., the signature must be verifiable using the SubjectPublicKeyInfo field);

* The subject and issuer fields **MUST** be identical;

* If the subject field is NULL, then both subjectAltNames and issuerAltNames extensions **MUST** be present and have exactly the same value;

* The values of all other extensions must be suitable for a self-signed certificate (e.g., key identifiers for subject and issuer must be the same).

```
OOBCertHash ::= SEQUENCE {
   hashAlg     [0] AlgorithmIdentifier     OPTIONAL,
   certId      [1] CertId                  OPTIONAL,
   hashVal         BIT STRING
}
```

The intention of the hash value is that anyone who has securely received the hash value (via the out-of-band means) can verify a self-signed certificate for that CA.

5.2.6. Archive Options

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

See [RFC4211] for PKIArchiveOptions syntax.

5.2.7. Publication Information

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

See [RFC4211] for PKIPublicationInfo syntax.

5.2.8. Proof-of-Possession Structures

< ToDo: This section should be aligned with Section 4.3 of this document and RFC 4211 Section 4. It should potentially be restructured and updated for better readability. Also some inconsistencies in Section 5.2.8.3 resulting from the update of RFC2510 to RFC4210 should be fixed. >

If the certification request is for a key pair that supports signing , then the proof-of-possession of the private signing key is demonstrated through use of the POPOSigningKey structure as defined in RFC 4211 [RFC4211].

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

POPOSigningKeyInput ::= SEQUENCE {
   authInfo             CHOICE {
      sender            [0] GeneralName,
      publicKeyMAC      PKMACValue
   },
   publicKey            SubjectPublicKeyInfo
}
```

The signature (using "algorithmIdentifier") is on the DER-encoded value of poposkInput (i.e., the "value" OCTETs of the POPOSigningKeyInput DER).

If certTemplate (or the altCertTemplate control as defined in Section 5.2.1) contains the subject and publicKey values, then poposkInput **MUST** be omitted and the signature **MUST** be computed on the DER-encoded value of certReq field if the CertReqMsg (or the DER-encoded value of AltCertTemplate). If certTemplate/ altCertTemplate does not contain both the subject and public key values (i.e., if it contains only one of these, or neither), then poposkInput **MUST** be present and **MUST** be signed.

On the other hand, if the certification request is for a key pair that does not support signing (i.e., a request for an encryption or KEM certificate), then the proof-of-possession of the private decryption key may be demonstrated in one of three ways, as detailed in the following subsections.

5.2.8.1. Inclusion of the Private Key

By the inclusion of the private key (encrypted) in the CertRequest (in the thisMessage field of POPOPrivKey or in the PKIArchiveOptions control structure, depending upon whether or not archival of the private key is also desired).

```
POPOPrivKey ::= CHOICE {
   thisMessage       [0] BIT STRING,   -- deprecated
   subsequentMessage [1] SubsequentMessage,
   dhMAC             [2] BIT STRING,   -- deprecated
   agreeMAC          [3] PKMACValue,
   encryptedKey      [4] EnvelopedData
}
```

Note: When using CMP V2 with EcryptedValue, RFC 4210 Appendix C [RFC4210] made the behavioral clarification of specifying that the contents of "thisMessage" **MUST** be encoded as an EncryptedValue and then wrapped in a BIT STRING. This allows the necessary conveyance and protection of the private key while maintaining bits-on-the-wire compatibility with RFC 4211 [RFC4211].

< ToDo: Section 2.27 of CMP Updated should be updated during AUTH48 specifying the use of encryptedKey field instead of thisMessage when EnvelopedData is used. >

5.2.8.2. Indirect Method

By having the CA return not the certificate, but an encrypted certificate (i.e., the certificate encrypted under a randomly-generated symmetric key, and the symmetric key encrypted under the public key for which the certification request is being made) -- this is the "indirect" method mentioned previously in Section 4.3.2. The end entity proves knowledge of the private decryption key to the CA by providing the correct CertHash for this certificate in the certConf message. This demonstrates POP because the EE can only compute the correct CertHash if it is able to recover the certificate, and it can only recover the certificate if it is able to decrypt the symmetric key using the required private key. Clearly, for this to work, the CA **MUST NOT** publish the certificate until the certConf message arrives (when certHash is to be used to demonstrate POP). See Section 5.3.18 for further details.

5.2.8.3. Challenge-Response Protocol

By having the end entity engage in a challenge-response protocol (using the messages POPODecKeyChall and POPODecKeyResp; see below) between CertReqMessages and CertRepMessage -- this is the "direct" method mentioned previously in Section 4.3.2. (This method would typically be used in an environment in which an RA verifies POP and then makes a certification request to the CA on behalf of the end entity. In such a scenario, the CA trusts the RA to have done POP correctly before the RA requests a certificate for the end entity.)

The complete protocol then looks as follows (note that req' does not necessarily encapsulate req as a nested message):

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

This protocol is obviously much longer than the 3-way exchange given in Section 5.2.8.2 above, but allows a local Registration Authority to be involved and has the property that the certificate itself is not actually created until the proof-of-possession is complete. In some environments, a different order of the above messages may be required, such as the following (this may be determined by policy):

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

If the cert. request is for a key agreement key (KAK) pair, then the POP can use any of the 3 ways described above for enc. key pairs, with the following changes: (1) the parenthetical text of Section 5.2.8.2 is replaced with "(i.e., the certificate encrypted under the symmetric key derived from the CA's private KAK and the public key for which the certification request is being made)"; (2) the first parenthetical text of the challenge field of "Challenge" below is replaced with "(using PreferredSymmAlg (see Section 5.3.19.4 and Appendix D.5) and a symmetric key derived from the CA's private KAK and the public key for which the certification request is being made)". Alternatively, the POP can use the POPOSigningKey structure given in [RFC4211] (where the alg field is DHBasedMAC and the signature field is the MAC) as a fourth alternative for demonstrating POP if the CA already has a D-H certificate that is known to the EE.

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

```
POPODecKeyChallContent ::= SEQUENCE OF Challenge

Challenge ::= SEQUENCE {
   owf                 AlgorithmIdentifier  OPTIONAL,
   witness             OCTET STRING,
   challenge           OCTET STRING
}

Rand ::= SEQUENCE {
   int                 INTEGER,
   sender              GeneralName
}
```

Note that the size of Rand needs to be appropriate for encryption under the public key of the requester. Given that "int" will typically not be longer than 64 bits, this leaves well over 100 bytes of room for the "sender" field when the modulus is 1024 bits. If, in some environment, names are so long that they cannot fit (e.g., very long DNs), then whatever portion will fit should be used (as long as it includes at least the common name, and as long as the receiver is able to deal meaningfully with the abbreviation).

```
POPODecKeyRespContent ::= SEQUENCE OF INTEGER
```

5.2.8.4. Summary of PoP Options

The text in this section provides several options with respect to POP techniques. Using "SK" for "signing key", "EK" for "encryption key", and "KAK" for "key agreement key", the techniques may be summarized as follows:

RAVerified;
SKPOP;
EKPOPThisMessage;
KAKPOPThisMessage;
KAKPOPThisMessageDHMAC;
EKPOPEncryptedCert;
KAKPOPEncryptedCert;
EKPOPChallengeResp; and
KAKPOPChallengeResp.

Given this array of options, it is natural to ask how an end entity can know what is supported by the CA/RA (i.e., which options it may use when requesting certificates). The following guidelines should clarify this situation for EE implementers.

RAVerified. This is not an EE decision; the RA uses this if and only if it has verified POP before forwarding the request on to the CA, so it is not possible for the EE to choose this technique.

SKPOP. If the EE has a signing key pair, this is the only POP method specified for use in the request for a corresponding certificate.

EKPOPThisMessage and KAKPOPThisMessage. Whether or not to give up its private key to the CA/RA is an EE decision. If the EE decides to reveal its key, then these are the only POP methods available in this specification to achieve this (and the key pair type will determine which of these two methods to use).

KAKPOPThisMessageDHMAC. The EE can only use this method if (1) the CA has a DH certificate available for this purpose, and (2) the EE already has a copy of this certificate. If both these conditions hold, then this technique is clearly supported and may be used by the EE, if desired.

EKPOPEncryptedCert, KAKPOPEncryptedCert, EKPOPChallengeResp, KAKPOPChallengeResp. The EE picks one of these (in the subsequentMessage field) in the request message, depending upon preference and key pair type. The EE is not doing POP at this point; it is simply indicating which method it wants to use. Therefore, if the CA/RA replies with a "badPOP" error, the EE can re-request using the other POP method chosen in subsequentMessage. Note, however, that this specification encourages the use of the EncryptedCert choice and, furthermore, says that the challenge-response would typically be used when an RA is involved and doing POP verification. Thus, the EE should be able to make an intelligent decision regarding which of these POP methods to choose in the request message.

< ToDo: Possibly add a section describing a POP mechanism for KEM keys. >

5.2.9. GeneralizedTime

GeneralizedTime is a standard ASN.1 type and **SHALL** be used as specified in RFC 5280 Section 4.1.2.5.2 [RFC5280].

5.3. Operation-Specific Data Structures

5.3.1. Initialization Request

An Initialization request message contains as the PKIBody a CertReqMessages data structure, which specifies the requested certificate(s). Typically, SubjectPublicKeyInfo, KeyId, and Validity are the template fields which may be supplied for each certificate requested (see the profiles defined in [RFCBBBB] Section 4.1.1, Appendix C.4 and Appendix D.7 for further information). This message is intended to be used for entities when first initializing into the PKI.

See Section 5.2.1 and [RFC4211] for CertReqMessages syntax.

5.3.2. Initialization Response

An Initialization response message contains as the PKIBody an CertRepMessage data structure, which has for each certificate requested a PKIStatusInfo field, a subject certificate, and possibly a private key (normally encrypted using EnvelopedData, see [RFCBBBB] Section 4.1.6 for further information).

See Section 5.3.4 for CertRepMessage syntax. Note that if the PKI Message Protection is "shared secret information" (see Section 5.1.3), then any certificate transported in the caPubs field may be directly trusted as a root CA certificate by the initiator.

5.3.3. Certification Request

A Certification request message contains as the PKIBody a CertReqMessages data structure, which specifies the requested certificates (see the profiles defined in [RFCBBBB] Section 4.1.2 and Appendix C.2 for further information). This message is intended to be used for existing PKI entities who wish to obtain additional certificates.

See Section 5.2.1 and [RFC4211] for CertReqMessages syntax.

Alternatively, the PKIBody **MAY** be a CertificationRequest (this structure is fully specified by the ASN.1 structure CertificationRequest given in [RFC2986], see the profiles defined in [RFCBBBB] Section 4.1.4 for further information). This structure may be required for certificate requests for signing key pairs when interoperation with legacy systems is desired, but its use is strongly discouraged whenever not absolutely necessary.

5.3.4. Certification Response

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

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

CertResponse ::= SEQUENCE {
   certReqId           INTEGER,
   status              PKIStatusInfo,
   certifiedKeyPair    CertifiedKeyPair     OPTIONAL,
   rspInfo             OCTET STRING         OPTIONAL
   -- analogous to the id-regInfo-utf8Pairs string defined
   -- for regInfo in CertReqMsg [RFC4211]
}

CertifiedKeyPair ::= SEQUENCE {
   certOrEncCert       CertOrEncCert,
   privateKey      [0] EncryptedKey         OPTIONAL,
   -- see [RFC4211] for comment on encoding
   publicationInfo [1] PKIPublicationInfo   OPTIONAL
}

CertOrEncCert ::= CHOICE {
   certificate     [0] CMPCertificate,
   encryptedCert   [1] EncryptedKey
}
```

A p10cr message contains exactly one CertificationRequestInfo data structure as specified in PKCS#10 [RFC2986] but no certReqId. Therefore, the certReqId in the corresponding certification response (cp) message **MUST** be set to -1.

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

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

The use of EncryptedKey is described in Section 5.2.2.

Note: To indicate support for EnvelopedData the pvno cmp2021 is introduced by this document. Details on the usage of different pvno values are described in Section 7.

5.3.5. Key Update Request Content

For key update requests the CertReqMessages syntax is used. Typically, SubjectPublicKeyInfo, KeyId, and Validity are the template fields that may be supplied for each key to be updated (see the profiles defined in [RFCBBBB] Section 4.1.3 and Appendix C.6 for further information). This message is intended to be used to request updates to existing (non-revoked and non-expired) certificates (therefore, it is sometimes referred to as a "Certificate Update" operation). An update is a replacement certificate containing either a new subject public key or the current subject public key (although the latter practice may not be appropriate for some environments).

SeeSection 5.2.1 and [RFC4211] for CertReqMessages syntax.

5.3.6. Key Update Response Content

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

See Section 5.3.4 for CertRepMessage syntax.

5.3.7. Key Recovery Request Content

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

See Section 5.2.1 and [RFC4211] for CertReqMessages syntax. Note that if a key history is required, the requester must supply a Protocol Encryption Key control in the request message.

5.3.8. Key Recovery Response Content

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

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

5.3.9. Revocation Request Content

When requesting revocation of a certificate (or several certificates), the following data structure is used (see the profiles defined in [RFCBBBB] Section 4.2 for further information). The name of the requester is present in the PKIHeader structure.

```
RevReqContent ::= SEQUENCE OF RevDetails

RevDetails ::= SEQUENCE {
   certDetails         CertTemplate,
   crlEntryDetails     Extensions       OPTIONAL
}
```

5.3.10. Revocation Response Content

The revocation response is the response to the above message. If produced, this is sent to the requester of the revocation. (A separate revocation announcement message **MAY** be sent to the subject of the certificate for which revocation was requested.)

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

5.3.11. Cross Certification Request Content

Cross certification requests use the same syntax (CertReqMessages) as normal certification requests, with the restriction that the key pair **MUST** have been generated by the requesting CA and the private key **MUST NOT** be sent to the responding CA (see the profiles defined in Appendix D.6 for further information). This request **MAY** also be used by subordinate CAs to get their certificates signed by the parent CA.

See Section 5.2.1 and [RFC4211] for CertReqMessages syntax.

5.3.12. Cross Certification Response Content

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

See Section 5.3.4 for CertRepMessage syntax.

5.3.13. CA Key Update Announcement Content

When a CA updates its own key pair, the following data structure **MAY** be used to announce this event.

```
CAKeyUpdAnnContent ::= SEQUENCE {
   oldWithNew         Certificate,
   newWithOld         Certificate,
   newWithNew         Certificate
}
```

5.3.14. Certificate Announcement

This structure **MAY** be used to announce the existence of certificates.

Note that this message is intended to be used for those cases (if any) where there is no pre-existing method for publication of certificates; it is not intended to be used where, for example, X.500 is the method for publication of certificates.

```
CertAnnContent ::= Certificate
```

5.3.15. Revocation Announcement

When a CA has revoked, or is about to revoke, a particular certificate, it **MAY** issue an announcement of this (possibly upcoming) event.

```
RevAnnContent ::= SEQUENCE {
   status              PKIStatus,
   certId              CertId,
   willBeRevokedAt     GeneralizedTime,
   badSinceDate        GeneralizedTime,
   crlDetails          Extensions  OPTIONAL
}
```

A CA **MAY** use such an announcement to warn (or notify) a subject that its certificate is about to be (or has been) revoked. This would typically be used where the request for revocation did not come from the subject concerned.

The willBeRevokedAt field contains the time at which a new entry will be added to the relevant CRLs.

5.3.16. CRL Announcement

When a CA issues a new CRL (or set of CRLs) the following data structure **MAY** be used to announce this event.

```
CRLAnnContent ::= SEQUENCE OF CertificateList
```

5.3.17. PKI Confirmation Content

This data structure is used in the protocol exchange as the final PKIMessage. Its content is the same in all cases -- actually there is no content since the PKIHeader carries all the required information.

```
PKIConfirmContent ::= NULL
```

Use of this message for certificate confirmation is **NOT RECOMMENDED**; certConf **SHOULD** be used instead. Upon receiving a PKIConfirm for a certificate response, the recipient **MAY** treat it as a certConf with all certificates being accepted.

5.3.18. Certificate Confirmation Content

This data structure is used by the client to send a confirmation to the CA/RA to accept or reject certificates.

```
CertStatus ::= SEQUENCE {
   certHash    OCTET STRING,
   certReqId   INTEGER,
   statusInfo  PKIStatusInfo OPTIONAL,
   hashAlg [0] AlgorithmIdentifier{DIGEST-ALGORITHM, {...}}
               OPTIONAL
}
```

The hashAlg field **SHOULD** be used only in exceptional cases where the signatureAlgorithm of the certificate to be confirmed does not specify a hash algorithm in the OID or in the parameters or does not define a hash algorithm to use with CMP, e.g., for EdDSA in [RFCCCCC] Section 3.3). Otherwise, the certHash value **SHALL** be computed using the same hash algorithm as used to create and verify the certificate signature. If hashAlg is used, the CMP version indicated by the certConf message header must be cmp2021(3).

For any particular CertStatus, omission of the statusInfo field indicates ACCEPTANCE of the specified certificate. Alternatively, explicit status details (with respect to acceptance or rejection) **MAY** be provided in the statusInfo field, perhaps for auditing purposes at the CA/RA.

Within CertConfirmContent, omission of a CertStatus structure corresponding to a certificate supplied in the previous response message indicates REJECTION of the certificate. Thus, an empty CertConfirmContent (a zero-length SEQUENCE) **MAY** be used to indicate rejection of all supplied certificates. See Section 5.2.8, item (2), for a discussion of the certHash field with respect to proof-of-possession.

5.3.19. PKI General Message Content

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

-- where {id-it} = {id-pkix 4} = {1 3 6 1 5 5 7 4}
GenMsgContent ::= SEQUENCE OF InfoTypeAndValue
```

5.3.19.1. CA Protocol Encryption Certificate

This **MAY** be used by the EE to get a certificate from the CA to use to protect sensitive information during the protocol.

```
GenMsg:    {id-it 1}, < absent >
GenRep:    {id-it 1}, Certificate | < absent >
```

EEs **MUST** ensure that the correct certificate is used for this purpose.

5.3.19.2. Signing Key Pair Types

This **MAY** be used by the EE to get the list of signature algorithm whose subject public key values the CA is willing to certify.

```
GenMsg:    {id-it 2}, < absent >
GenRep:    {id-it 2}, SEQUENCE SIZE (1..MAX) OF
                        AlgorithmIdentifier
```

Note: For the purposes of this exchange, rsaEncryption and rsaWithSHA1, for example, are considered to be equivalent; the question being asked is, "Is the CA willing to certify an RSA public key?"

Note: In case several EC curves are supported, several id-ecPublicKey elements as defined in RFC 5480 [RFC5480] need to be given, one per named curve.

5.3.19.3. Encryption/Key Agreement Key Pair Types

This **MAY** be used by the client to get the list of encryption/key agreement algorithms whose subject public key values the CA is willing to certify.

```
GenMsg:    {id-it 3}, < absent >
GenRep:    {id-it 3}, SEQUENCE SIZE (1..MAX) OF
                          AlgorithmIdentifier
```

Note: In case several EC curves are supported, several id-ecPublicKey elements as defined in RFC 5480 [RFC5480] need to be given, one per named curve.

5.3.19.4. Preferred Symmetric Algorithm

This **MAY** be used by the client to get the CA-preferred symmetric encryption algorithm for any confidential information that needs to be exchanged between the EE and the CA (for example, if the EE wants to send its private decryption key to the CA for archival purposes).

```
GenMsg:    {id-it 4}, < absent >
GenRep:    {id-it 4}, AlgorithmIdentifier
```

5.3.19.5. Updated CA Key Pair

This **MAY** be used by the CA to announce a CA key update event.

```
GenMsg:    {id-it 5}, CAKeyUpdAnnContent
```

5.3.19.6. CRL

This **MAY** be used by the client to get a copy of the latest CRL.

```
GenMsg:    {id-it 6}, < absent >
GenRep:    {id-it 6}, CertificateList
```

5.3.19.7. Unsupported Object Identifiers

This is used by the server to return a list of object identifiers that it does not recognize or support from the list submitted by the client.

```
GenRep:    {id-it 7}, SEQUENCE SIZE (1..MAX) OF OBJECT IDENTIFIER
```

5.3.19.8. Key Pair Parameters

This **MAY** be used by the EE to request the domain parameters to use for generating the key pair for certain public-key algorithms. It can be used, for example, to request the appropriate P, Q, and G to generate the DH/DSA key, or to request a set of well-known elliptic curves.

```
GenMsg:    {id-it 10}, OBJECT IDENTIFIER -- (Algorithm object-id)
GenRep:    {id-it 11}, AlgorithmIdentifier | < absent >
```

An absent infoValue in the GenRep indicates that the algorithm specified in GenMsg is not supported.

EEs **MUST** ensure that the parameters are acceptable to it and that the GenRep message is authenticated (to avoid substitution attacks).

5.3.19.9. Revocation Passphrase

This **MAY** be used by the EE to send a passphrase to a CA/RA for the purpose of authenticating a later revocation request (in the case that the appropriate signing private key is no longer available to authenticate the request). See Appendix B for further details on the use of this mechanism.

```
GenMsg:    {id-it 12}, EncryptedKey
GenRep:    {id-it 12}, < absent >
```

The use of EncryptedKey is described in Section 5.2.2.

5.3.19.10. ImplicitConfirm

See Section 5.1.1.1 for the definition and use of {id-it 13}.

5.3.19.11. ConfirmWaitTime

See Section 5.1.1.2 for the definition and use of {id-it 14}.

5.3.19.12. Original PKIMessage

See Section 5.1.1.3 for the definition and use of {id-it 15}.

5.3.19.13. Supported Language Tags

This **MAY** be used to determine the appropriate language tag to use in subsequent messages. The sender sends its list of supported languages (in order, most preferred to least); the receiver returns the one it wishes to use. (Note: each UTF8String **MUST** include a language tag.) If none of the offered tags are supported, an error **MUST** be returned.

```
GenMsg:    {id-it 16}, SEQUENCE SIZE (1..MAX) OF UTF8String
GenRep:    {id-it 16}, SEQUENCE SIZE (1) OF UTF8String
```

5.3.19.14. CA Certificates

This **MAY** be used by the client to get CA certificates.

```
GenMsg:    {id-it 17}, < absent >
GenRep:    {id-it 17}, SEQUENCE SIZE (1..MAX) OF
                         CMPCertificate | < absent >
```

5.3.19.15. Root CA Update

This **MAY** be used by the client to get an update of a root CA certificate, which is provided in the body of the request message. In contrast to the ckuann message this approach follows the request/response model.

```
GenMsg:    {id-it 20}, RootCaCertValue | < absent >
GenRep:    {id-it 18}, RootCaKeyUpdateContent | < absent >

RootCaCertValue ::= CMPCertificate

RootCaKeyUpdateValue ::= RootCaKeyUpdateContent

RootCaKeyUpdateContent ::= SEQUENCE {
   newWithNew              CMPCertificate,
   newWithOld          [0] CMPCertificate OPTIONAL,
   oldWithNew          [1] CMPCertificate OPTIONAL
}
```

Note: In contrast to CAKeyUpdAnnContent, this type offers omitting newWithOld and oldWithNew in the GenRep message, depending on the needs of the EE.

5.3.19.16. Certificate Request Template

This **MAY** be used by the client to get a template containing requirements for certificate request attributes and extensions. The controls id-regCtrl-algId and id-regCtrl-rsaKeyLen **MAY** contain details on the types of subject public keys the CA is willing to certify.

The id-regCtrl-algId control **MAY** be used to identify a cryptographic algorithm, see RFC 5280 Section 4.1.2.7 [RFC5280], other than rsaEncryption. The algorithm field **SHALL** identify a cryptographic algorithm. The contents of the optional parameters field will vary according to the algorithm identified. For example, when the algorithm is set to id-ecPublicKey, the parameters identify the elliptic curve to be used, see [RFC5480].

Note: The client may specify a profile name in the certProfile field, see Section 5.1.1.4.

The id-regCtrl-rsaKeyLen control **SHALL** be used for algorithm rsaEncryption and **SHALL** contain the intended modulus bit length of the RSA key.

```
GenMsg:    {id-it 19}, < absent >
GenRep:    {id-it 19}, CertReqTemplateContent | < absent >

CertReqTemplateValue  ::= CertReqTemplateContent

CertReqTemplateContent ::= SEQUENCE {
   certTemplate           CertTemplate,
   keySpec                Controls OPTIONAL }

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

id-regCtrl-algId OBJECT IDENTIFIER ::= { iso(1)
   identified-organization(3) dod(6) internet(1) security(5)
   mechanisms(5) pkix(7) pkip(5) regCtrl(1) 11 }

AlgIdCtrl ::= AlgorithmIdentifier{ALGORITHM, {...}}

id-regCtrl-rsaKeyLen OBJECT IDENTIFIER ::= { iso(1)
   identified-organization(3) dod(6) internet(1) security(5)
   mechanisms(5) pkix(7) pkip(5) regCtrl(1) 12 }

RsaKeyLenCtrl ::= INTEGER (1..MAX)
```

The CertReqTemplateValue contains the prefilled certTemplate to be used for a future certificate request. The publicKey field in the certTemplate **MUST NOT** be used. In case the PKI management entity wishes to specify supported public-key algorithms, the keySpec field **MUST** be used. One AttributeTypeAndValue per supported algorithm or RSA key length **MUST** be used.

Note: The Controls ASN.1 type is defined in CRMF Section 6 [RFC4211]

5.3.19.17. CRL Update Retrieval

This **MAY** be used by the client to get new CRLs, specifying the source of the CRLs and the thisUpdate value of the latest CRL it already has, if available. A CRL source is given either by a DistributionPointName or the GeneralNames of the issuing CA. The DistributionPointName should be treated as an internal pointer to identify a CRL that the server already has and not as a way to ask the server to fetch CRLs from external locations. The server **SHALL** provide only those CRLs that are more recent than the ones indicated by the client.

```
GenMsg:    {id-it 22}, SEQUENCE SIZE (1..MAX) OF CRLStatus
GenRep:    {id-it 23}, SEQUENCE SIZE (1..MAX) OF
                           CertificateList  |  < absent >

CRLSource ::= CHOICE {
   dpn          [0] DistributionPointName,
   issuer       [1] GeneralNames }

CRLStatus ::= SEQUENCE {
   source       CRLSource,
   thisUpdate   Time OPTIONAL }
```

5.3.20. PKI General Response Content

```
GenRepContent ::= SEQUENCE OF InfoTypeAndValue
```

Examples of GenReps that **MAY** be supported include those listed in the subsections of Section 5.3.19.

5.3.21. Error Message Content

This data structure **MAY** be used by EE, CA, or RA to convey error info and by a PKI management entity to initiate delayed delivery of responses.

```
ErrorMsgContent ::= SEQUENCE {
   pKIStatusInfo          PKIStatusInfo,
   errorCode              INTEGER           OPTIONAL,
   errorDetails           PKIFreeText       OPTIONAL
}
```

This message **MAY** be generated at any time during a PKI transaction. If the client sends this request, the server **MUST** respond with a PKIConfirm response, or another ErrorMsg if any part of the header is not valid.

In case a PKI management entity sends an error message to the EE with the pKIStatusInfo field containing the status "waiting", the EE **SHOULD** initiate polling as described in Section 5.3.22. If the EE does not initiate polling, both sides **MUST** treat this message as the end of the transaction (if a transaction is in progress).

If protection is desired on the message, the client **MUST** protect it using the same technique (i.e., signature or MAC) as the starting message of the transaction. The CA **MUST** always sign it with a signature key.

5.3.22. Polling Request and Response

This pair of messages is intended to handle scenarios in which the client needs to poll the server to determine the status of an outstanding response (i.e., when the "waiting" PKIStatus has been received).

```
PollReqContent ::= SEQUENCE OF SEQUENCE {
   certReqId    INTEGER }

PollRepContent ::= SEQUENCE OF SEQUENCE {
   certReqId    INTEGER,
   checkAfter   INTEGER,  -- time in seconds
   reason       PKIFreeText OPTIONAL }
```

In response to an ir, cr, p10cr, or kur request message, polling is initiated with an ip, cp, or kup response message containing status "waiting". For any type of request message, polling can be initiated with an error response messages with status "waiting". The following clauses describe how polling messages are used. It is assumed that multiple certConf messages can be sent during transactions. There will be one sent in response to each ip, cp, or kup that contains a CertStatus for an issued certificate.

1 In response to an ip, cp, or kup message, an EE will send a certConf for all issued certificates and expect a PKIconf for each certConf. An EE will send a pollReq message in response to each CertResponse element of an ip, cp, or kup message with status "waiting" and in response to an error message with status "waiting". Its certReqId **MUST** be either the index of a CertResponse data structure with status "waiting" or -1 referring to the complete response.

2 In response to a pollReq, a CA/RA will return an ip, cp, or kup if one or more of still pending requested certificates are ready or the final response to some other type of request is available; otherwise, it will return a pollRep.

3 If the EE receives a pollRep, it will wait for at least the number of seconds given in the checkAfter field before sending another pollReq.

4 If the EE receives an ip, cp, or kup, then it will be treated in the same way as the initial response; if it receives any other response, then this will be treated as the final response to the original request.

The following client-side state machine describes polling for individual CertResponse elements.

```
                            START
                              |
                              v
                           Send ir
                              | ip
                              v
                         Check status
                         of returned <------------------------+
                            certs                             |
                              |                               |
    +------------------------>|<------------------+           |
    |                         |                   |           |
    |        (issued)         v       (waiting)   |           |
  Add to <----------- Check CertResponse ------> Add to       |
 conf list           for each certificate      pending list   |
                              /                               |
                             /                                |
                (conf list) /     (empty conf list)           |
                           /                     ip           |
                          /                 +-----------------+
   (empty pending list)  /                  |    pollRep
     END <---- Send certConf        Send pollReq---------->Wait
                      |                 ^   ^               |
                      |                 |   |               |
                      +-----------------+   +---------------+
                         (pending list)
```

In the following exchange, the end entity is enrolling for two certificates in one request.

```
Step  End Entity                       PKI
--------------------------------------------------------------------
1   Format ir
2                    -> ir      ->
3                                    Handle ir
4                                    Manual intervention is
                                     required for both certs.
5                    <- ip      <-
6   Process ip
7   Format pollReq
8                    -> pollReq  ->
9                                    Check status of cert requests
10                                   Certificates not ready
11                                   Format pollRep
12                   <- pollRep  <-
13  Wait
14  Format pollReq
15                   -> pollReq  ->
16                                   Check status of cert requests
17                                   One certificate is ready
18                                   Format ip
19                   <- ip       <-
20  Handle ip
21  Format certConf
22                   -> certConf ->
23                                   Handle certConf
24                                   Format ack
25                   <- pkiConf   <-
26  Format pollReq
27                   -> pollReq  ->
28                                   Check status of certificate
29                                   Certificate is ready
30                                   Format ip
31                   <- ip       <-
31  Handle ip
32  Format certConf
33                   -> certConf ->
34                                   Handle certConf
35                                   Format ack
36                   <- pkiConf  <-
```

The following client-side state machine describes polling for a complete response message.

```
                                Start
                                  |
                                  | Send request
                                  |
             +----------- Receive response ------------+
             |                                         |
             | ip/cp/kup/error with                    | other
             | status "waiting"                        | response
             |                                         |
             v                                         |
 +------> Polling                                      |
 |           |                                         |
 |           | Send pollReq                            |
 |           | Receive response                        |
 |           |                                         |
 |   pollRep | other response                          |
 +-----------+------------------->+<-------------------+
                                  |
                                  v
                            Handle response
                                  |
                                  v
                                 End
```

In the following exchange, the end entity is sending a general message request, and the response is delayed by the server.

```
Step  End Entity                       PKI
--------------------------------------------------------------------
1   Format genm
2                  -> genm     ->
3                                   Handle genm
4                                   delay in response is necessary
5                                   Format error message "waiting"
                                      with certReqId set to -1
6                   <- error   <-
7   Process error
8   Format pollReq
9                   -> pollReq ->
10                                  Check status of original request
                                    general message response not ready
11                                  Format pollRep
12                  <- pollRep <-
13  Wait
14  Format pollReq
15                  -> pollReq ->
16                                  Check status of original request
                                    general message response is ready
17                                  Format genp
18                  <- genp    <-
19  Handle genp
```

6. Mandatory PKI Management Functions

Some of the PKI management functions outlined in Section 3.1 above are described in this section.

This section deals with functions that are "mandatory" in the sense that all end entity and CA/RA implementations **MUST** be able to provide the functionality described. This part is effectively the profile of the PKI management functionality that **MUST** be supported. Note, however, that the management functions described in this section do not need to be accomplished using the PKI messages defined in Section 5 if alternate means are suitable for a given environment (see [RFCBBBB] Section 7 and Appendix C for profiles of the PKIMessages that **MUST** be supported).

6.1. Root CA Initialization

[See Section 3.1.1.2 for this document's definition of "root CA".]

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

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

The data structure used to carry the fingerprint is the OOBCertHash, see Section 5.2.5.

6.2. Root CA Key Update

CA keys (as all other keys) have a finite lifetime and will have to be updated on a periodic basis. The certificates NewWithNew, NewWithOld, and OldWithNew (see Section 4.4.1) **MAY** be issued by the CA to aid existing end entities who hold the current self-signed CA certificate (OldWithOld) to transition securely to the new self-signed CA certificate (NewWithNew), and to aid new end entities who will hold NewWithNew to acquire OldWithOld securely for verification of existing data.

6.3. Subordinate CA Initialization

[See Section 3.1.1.2 for this document's definition of "subordinate CA".]

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

6.4. CRL production

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

6.5. PKI Information Request

When a PKI entity (CA, RA, or EE) wishes to acquire information about the current status of a CA, it **MAY** send that CA a request for such information.

The CA **MUST** respond to the request by providing (at least) all of the information requested by the requester. If some of the information cannot be provided, then an error must be conveyed to the requester.

If PKIMessages are used to request and supply this PKI information, then the request **MUST** be the GenMsg message, the response **MUST** be the GenRep message, and the error **MUST** be the Error message. These messages are protected using a MAC based on shared secret information (i.e., password-based MAC, see CMP Algorithms [RFCCCCC] Section 6.1) or a signature(if the end entity has an existing certificate).

6.6. Cross Certification

The requester CA is the CA that will become the subject of the cross-certificate; the responder CA will become the issuer of the cross-certificate.

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

6.6.1. One-Way Request-Response Scheme:

The cross-certification scheme is essentially a one way operation; that is, when successful, this operation results in the creation of one new cross-certificate. If the requirement is that cross-certificates be created in "both directions", then each CA, in turn, must initiate a cross-certification operation (or use another scheme).

This scheme is suitable where the two CAs in question can already verify each other's signatures (they have some common points of trust) or where there is an out-of-band verification of the origin of the certification request.

Detailed Description:

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

The authorization code is used for authentication and integrity purposes. This is done by generating a symmetric key based on the authorization code and using the symmetric key for generating Message Authentication Codes (MACs) on all messages exchanged. (Authentication may alternatively be done using signatures instead of MACs, if the CAs are able to retrieve and validate the required public keys by some means, such as an out-of-band hash comparison.)

The requester CA initiates the exchange by generating a cross-certification request (ccr) with a fresh random number (requester random number). The requester CA then sends the ccr message to the responder CA. The fields in this message are protected from modification with a MAC based on the authorization code.

Upon receipt of the ccr message, the responder CA validates the message and the MAC, saves the requester random number, and generates its own random number (responder random number). It then generates (and archives, if desired) a new requester certificate that contains the requester CA public key and is signed with the responder CA signature private key. The responder CA responds with the cross certification response (ccp) message. The fields in this message are protected from modification with a MAC based on the authorization code.

Upon receipt of the ccp message, the requester CA validates the message (including the received random numbers) and the MAC. The requester CA responds with the certConf message. The fields in this message are protected from modification with a MAC based on the authorization code. The requester CA **MAY** write the requester certificate to the Repository as an aid to later certificate path construction.

Upon receipt of the certConf message, the responder CA validates the message and the MAC, and sends back an acknowledgement using the PKIConfirm message. It **MAY** also publish the requester certificate as an aid to later path construction.

Notes:

1. The ccr message must contain a "complete" certification request; that is, all fields except the serial number (including, e.g., a BasicConstraints extension) must be specified by the requester CA.

2. The ccp message **SHOULD** contain the verification certificate of the responder CA; if present, the requester CA must then verify this certificate (for example, via the "out-of-band" mechanism).

(A simpler, non-interactive model of cross-certification may also be envisioned, in which the issuing CA acquires the subject CA's public key from some repository, verifies it via some out-of-band mechanism, and creates and publishes the cross-certificate without the subject CA's explicit involvement. This model may be perfectly legitimate for many environments, but since it does not require any protocol message exchanges, its detailed description is outside the scope of this specification.)

6.7. End Entity Initialization

As with CAs, end entities must be initialized. Initialization of end entities requires at least two steps:

- acquisition of PKI information

- out-of-band verification of one root-CA public key

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

6.7.1. Acquisition of PKI Information

The information **REQUIRED** is:

- the current root-CA public key

- (if the certifying CA is not a root-CA) the certification path from the root CA to the certifying CA together with appropriate revocation lists

- the algorithms and algorithm parameters that the certifying CA supports for each relevant usage

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

The required information **MAY** be acquired as described in Section 6.5.

6.7.2. Out-of-Band Verification of Root-CA Key

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

See Section 6.1 for further details.

6.8. Certificate Request

An initialized end entity **MAY** request an additional certificate at any time (for any purpose). This request will be made using the certification request (cr) message. If the end entity already possesses a signing key pair (with a corresponding verification certificate), then this cr message will typically be protected by the entity's digital signature. The CA returns the new certificate (if the request is successful) in a CertRepMessage.

6.9. Key Update

When a key pair is due to expire, the relevant end entity **MAY** request a key update; that is, it **MAY** request that the CA issue a new certificate for a new key pair (or, in certain circumstances, a new certificate for the same key pair). The request is made using a key update request (kur) message (referred to, in some environments, as a "Certificate Update" operation). If the end entity already possesses a signing key pair (with a corresponding verification certificate), then this message will typically be protected by the entity's digital signature. The CA returns the new certificate (if the request is successful) in a key update response (kup) message, which is syntactically identical to a CertRepMessage.

7. Version Negotiation

This section defines the version negotiation used to support older protocols between client and servers.

If a client knows the protocol version(s) supported by the server (e.g., from a previous PKIMessage exchange or via some out-of-band means), then it **MUST** send a PKIMessage with the highest version supported by both it and the server. If a client does not know what version(s) the server supports, then it **MUST** send a PKIMessage using the highest version it supports, with the following exception. Version cmp2021 **SHOULD** only be used if cmp2021 syntax is needed for the request being sent or for the expected response.

Note: Using cmp2000 as the default pvno is done to avoid extra message exchanges for version negotiation and to foster compatibility with cmp2000 implementations. Version cmp2021 syntax is only needed if a message exchange uses hashAlg (in CertStatus) or EnvelopedData.

If a server receives a message with a version that it supports, then the version of the response message **MUST** be the same as the received version. If a server receives a message with a version higher or lower than it supports, then it **MUST** send back an ErrorMsg with the unsupportedVersion bit set (in the failureInfo field of the pKIStatusInfo). If the received version is higher than the highest supported version, then the version in the error message **MUST** be the highest version the server supports; if the received version is lower than the lowest supported version then the version in the error message **MUST** be the lowest version the server supports.

If a client gets back an ErrorMsgContent with the unsupportedVersion bit set and a version it supports, then it **MAY** retry the request with that version.

7.1. Supporting RFC 2510 Implementations

RFC 2510 did not specify the behaviour of implementations receiving versions they did not understand since there was only one version in existence. With the introduction of the revision in [RFC4210], the following versioning behaviour is recommended.

7.1.1. Clients Talking to RFC 2510 Servers

If, after sending a message with a protocol version number higher than cmp1999, a client receives an ErrorMsgContent with a version of cmp1999, then it **MUST** abort the current transaction.

If a client receives a non-error PKIMessage with a version of cmp1999, then it **MAY** decide to continue the transaction (if the transaction hasn't finished) using RFC 2510 semantics. If it does not choose to do so and the transaction is not finished, then it **MUST** abort the transaction and send an ErrorMsgContent with a version of cmp1999.

7.1.2. Servers Receiving Version cmp1999 PKIMessages

If a server receives a version cmp1999 message it **MAY** revert to RFC 2510 behaviour and respond with version cmp1999 messages. If it does not choose to do so, then it **MUST** send back an ErrorMsgContent as described above in Section 7.

8. Security Considerations

8.1. Proof-Of-Possession with a Decryption Key

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

8.2. Proof-Of-Possession by Exposing the Private Key

Note also that exposing a private key to the CA/RA as a proof-of-possession technique can carry some security risks (depending upon whether or not the CA/RA can be trusted to handle such material appropriately). Implementers are advised to:

* Exercise caution in selecting and using this particular POP mechanism

* When appropriate, have the user of the application explicitly state that they are willing to trust the CA/RA to have a copy of their private key before proceeding to reveal the private key.

8.3. Attack Against Diffie-Hellman Key Exchange

A small subgroup attack during a Diffie-Hellman key exchange may be carried out as follows. A malicious end entity may deliberately choose D-H parameters that enable him/her to derive (a significant number of bits of) the D-H private key of the CA during a key archival or key recovery operation. Armed with this knowledge, the EE would then be able to retrieve the decryption private key of another unsuspecting end entity, EE2, during EE2's legitimate key archival or key recovery operation with that CA. In order to avoid the possibility of such an attack, two courses of action are available. (1) The CA may generate a fresh D-H key pair to be used as a protocol encryption key pair for each EE with which it interacts. (2) The CA may enter into a key validation protocol (not specified in this document) with each requesting end entity to ensure that the EE's protocol encryption key pair will not facilitate this attack. Option (1) is clearly simpler (requiring no extra protocol exchanges from either party) and is therefore **RECOMMENDED**.

8.4. Private Keys for Certificate Signing and CMP Message Protection

A CA should not reuse its certificate signing key for other purposes such as protecting CMP responses and TLS connections. This way, exposure to other parts of the system and the number of uses of this particularly critical key is reduced to a minimum.

8.5. Entropy of Random Numbers, Key Pairs, and Shared Secret Information

Implementations must generate nonces and private keys from random input. The use of inadequate pseudo-random number generators (PRNGs) to generate cryptographic keys can result in little or no security. An attacker may find it much easier to reproduce the PRNG environment that produced the keys and to search the resulting small set of possibilities than brute-force searching the whole key space. As an example of predictable random numbers see [CVE-2008-0166]; consequences of low-entropy random numbers are discussed in Mining Your Ps and Qs [MiningPsQs]. The generation of quality random numbers is difficult. ISO/IEC 20543:2019 [ISO.20543-2019], NIST SP 800-90A Rev.1 [NIST.SP.800_90Ar1], BSI AIS 31 V2.0 [AIS31], and others offer valuable guidance in this area.

If shared secret information is generated by a cryptographically secure random-number generator (CSRNG) it is safe to assume that the entropy of the shared secret information equals its bit length. If no CSRNG is used, the entropy of a shared secret information depends on the details of the generation process and cannot be measured securely after it has been generated. If user-generated passwords are used as shared secret information, their entropy cannot be measured and are typically insufficient for protected delivery of centrally generated keys or trust anchors.

If the entropy of a shared secret information protecting the delivery of a centrally generated key pair is known, it should not be less than the security strength of that key pair; if the shared secret information is re-used for different key pairs, the security of the shared secret information should exceed the security strength of each individual key pair.

For the case of a PKI management operation that delivers a new trust anchor (e.g., a root CA certificate) using caPubs or genm (a) that is not concluded in a timely manner or (b) where the shared secret information is re-used for several key management operations, the entropy of the shared secret information, if known, should not be less than the security strength of the trust anchor being managed by the operation. The shared secret information should have an entropy that at least matches the security strength of the key material being managed by the operation. Certain use cases may require shared secret information that may be of a low security strength, e.g., a human generated password. It is **RECOMMENDED** that such secret information be limited to a single PKI management operation.

Importantly for this section further information about algorithm use profiles and their security strength is available in CMP Algorithms [RFCCCC] Section 7.

8.6. Trust Anchor Provisioning Using CMP Messages

A provider of trust anchors, which may be an RA involved in configuration management of its clients, **MUST NOT** include to-be-trusted CA certificates in a CMP message unless the specific deployment scenario can ensure that it is adequate that the receiving EE trusts these certificates, e.g., by loading them into its trust store.

Whenever an EE receives in a CMP message, e.g., in the caPubs field of a certificate response or in a general response (genp), a CA certificate for use as a trust anchor, it **MUST** properly authenticate the message sender with existing trust anchors without requiring new trust anchors included in the message.

Additionally, the EE **MUST** verify that the sender is an authorized source of trust anchors. This authorization is governed by local policy and typically indicated using shared secret information or with a signature-based message protection using a certificate issued by a PKI that is explicitly authorized for this purpose.

8.7. Authorizing Requests for Certificates with Specific EKUs

When a CA issues a certificate containing extended key usage extensions as defined in Section 4.5, this expresses delegation of an authorization that originally is only with the CA certificate itself. Such delegation is a very sensitive action in a PKI and therefore special care must be taken when approving such certificate requests to ensure that only legitimate entities receive a certificate containing such an EKU.

8.8. Combiner Function for Hybrid Key Encapsulation Mechanisms

[I-D.ounsworth-cfrg-kem-combiners] presents the KEM combiner KDF(concat(H(ss1), H(ss2))) for suitable choices of a key derivation function KDF and a cryptographic hash function H. It argues that this construction can safely be reduced to KDF(concat(ss1, ss2)) when the KEMs being combined already include a KDF as part of computing their output. The dual KEM construction presented in Section 5.1.3.2 conforms to this construction in the following way. The output of the first HPKE, ss1 is computed via ReceiveExportBase(..) (RFC 9180 section 6.2 [RFC9180]), which chains to Context.Export(..) (RFC 9180 section 5.3 [RFC9180]), which chains to LabeledExpand(..) and Expand(..) (RFC 9180 section 4 [RFC9180]), which uses a KDF to expand the KEM output prk into ss1. That matches or exceeds the security strength of H(ss1) from [I-D.ounsworth-cfrg-kem-combiners]. Next, the dual KEM construction presented in Section 5.1.3.1 uses the HPKE output ss1 as part of the label context2 for the second HPKE. This in turn is passed through

ReceiveExportBase(..), Context.Export(..), LabeledExpand(..), and Expand(..) as above where finally the label context2, which is the info parameter for the Export(prk, info, l) function, and which contains the output of the first HPKE ss1, is concatenated with the output of the second KEM prk to produce the final shared secret ss2. That means the dual KEM construction defined in Section 5.1.3.1 maps to the notation of [I-D.ounsworth-cfrg-kem-combiners] as KDF(concat(ss2, KDF(ss1)), which is cryptographically stronger than the combiner KDF(ss1 || ss2) so long as the underlying KEM used in the second HPKE uses internally a KDF for deriving its output.

8.9. Usage of Certificate Transparency Logs

CAs that support indirect POP **MUST NOT** also publish final certificates to Certificate Transparency logs [RFC9162] before having received the certConf message containing the certHash of that certificate to complete the POP. The risk is that a malicious actor could fetch the final certificate from the CT log and use that to spoof a response to the implicit POP challenge via a certConf response. This risk does not apply to CT precertificates, so those are ok to publish.

If a certificate or its precertificate was published in a CT log it must be revoked, if a required certConf message could not be verified, especially when the implicit POP was used.

9. IANA Considerations

The IANA has already registered what is specified in CMP Updates [RFCAAAA].

No further action by the IANA is necessary for this document or any anticipated updates.

10. Acknowledgements

The authors of this document wish to thank Carlisle Adams, Stephen Farrell, Tomi Kause, and Tero Mononen, the original authors of [RFC4210], for their work.

We also thank all reviewers of this document for their valuable feedback.

11. References

11.1. Normative References

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, DOI

10.17487/RFC2985, November 2000, <https://www.rfc-editor.org/rfc/rfc2985>.

[RFC2986] 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>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/rfc/rfc3629>.

[RFC4211] 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>.

[RFC5280] 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>.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <https://www.rfc-editor.org/rfc/rfc5480>.

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

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <https://www.rfc-editor.org/rfc/rfc5958>.

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

[RFC8933] Housley, R., "Update to the Cryptographic Message Syntax (CMS) for Algorithm Identifier Protection", RFC 8933, DOI

10.17487/RFC8933, October 2020, <https://www.rfc-editor.org/rfc/rfc8933>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/rfc/rfc9180>.

[ITU.X509.2000] International Telecommunications Union, "Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks", ITU-T Recommendation X.509, ISO Standard 9594-8, March 2000.

[I-D.ietf-lamps-cmp-algorithms] Brockhaus, H., Aschauer, H., Ounsworth, M., and J. Gray, "Certificate Management Protocol (CMP) Algorithms", Work in Progress, Internet-Draft, draft-ietf-lamps-cmp-algorithms-15, 2 June 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-cmp-algorithms-15>.

[I-D.ietf-lamps-cms-kemri] Housley, R., Gray, J., and T. Okubo, "Using Key Encapsulation Mechanism (KEM) Algorithms in the Cryptographic Message Syntax (CMS)", Work in Progress, Internet-Draft, draft-ietf-lamps-cms-kemri-00, 24 February 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-cms-kemri-00>.

[MvOV97] Menezes, A., van Oorschot, P., and S. Vanstone, "Handbook of Applied Cryptography", CRC Press ISBN 0-8493-8523-7, 1996.

[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>.

11.2. Informative References

[I-D.ietf-lamps-cmp-updates] Brockhaus, H., von Oheimb, D., and J. Gray, "Certificate Management Protocol (CMP) Updates", Work in Progress, Internet-Draft, draft-ietf-lamps-cmp-updates-23, 29 June 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-cmp-updates-23>.

[I-D.ietf-lamps-lightweight-cmp-profile] Brockhaus, H., von Oheimb, D., and S. Fries, "Lightweight Certificate Management

Protocol (CMP) Profile", Work in Progress, Internet-Draft, draft-ietf-lamps-lightweight-cmp-profile-21, 17 February 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-lightweight-cmp-profile-21>.

[I-D.ietf-ace-cmpv2-coap-transport] Sahni, M. and S. Tripathi, "CoAP Transfer for the Certificate Management Protocol", Work in Progress, Internet-Draft, draft-ietf-ace-cmpv2-coap-transport-07, 27 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-cmpv2-coap-transport-07>.

[I-D.ietf-lamps-rfc6712bis] Brockhaus, H., von Oheimb, D., Ounsworth, M., and J. Gray, "Internet X.509 Public Key Infrastructure -- HTTP Transfer for the Certificate Management Protocol (CMP)", Work in Progress, Internet-Draft, draft-ietf-lamps-rfc6712bis-03, 10 February 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-rfc6712bis-03>.

[I-D.ounsworth-cfrg-kem-combiners] Ounsworth, M., Wussler, A., and S. Kousidis, "Combiner function for hybrid key encapsulation mechanisms (Hybrid KEMs)", Work in Progress, Internet-Draft, draft-ounsworth-cfrg-kem-combiners-03, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ounsworth-cfrg-kem-combiners-03>.

[RFC1847] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, DOI 10.17487/RFC1847, October 1995, <https://www.rfc-editor.org/rfc/rfc1847>.

[RFC2510] Adams, C. and S. Farrell, "Internet X.509 Public Key Infrastructure Certificate Management Protocols", RFC 2510, DOI 10.17487/RFC2510, March 1999, <https://www.rfc-editor.org/rfc/rfc2510>.

[RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <https://www.rfc-editor.org/rfc/rfc4210>.

[RFC4510] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, DOI

10.17487/RFC4510, June 2006, <https://www.rfc-editor.org/rfc/rfc4510>.

[RFC5912] 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>.

[RFC8649] Housley, R., "Hash Of Root Key Certificate Extension", RFC 8649, DOI 10.17487/RFC8649, August 2019, <https://www.rfc-editor.org/rfc/rfc8649>.

[RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, DOI 10.17487/RFC9162, December 2021, <https://www.rfc-editor.org/rfc/rfc9162>.

[NIST.SP.800_90Ar1] Barker, E. B., Kelsey, J. M., and NIST, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators", NIST Special Publications (General) 800-90Ar1, DOI 10.6028/NIST.SP.800-90Ar1, June 2015, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf>.

[IEEE.802.1AR-2018] IEEE, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", IEEE 802.1AR-2018, DOI 10.1109/IEEESTD.2018.8423794, August 2018, <https://ieeexplore.ieee.org/document/8423794>.

[CVE-2008-0166] National Institute of Science and Technology (NIST), "National Vulnerability Database - CVE-2008-0166", May 2008, <https://nvd.nist.gov/vuln/detail/CVE-2008-0166>.

[MiningPsQs] Security'12: Proceedings of the 21st USENIX conference on Security symposium, Heninger, N., Durumeric, Z., Wustrow, E., and J. A. Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices", August 2012, <https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger>.

[ISO.20543-2019] International Organization for Standardization (ISO), "Information technology -- Security techniques -- Test and analysis methods for random bit generators within ISO/IEC 19790 and ISO/IEC 15408", ISO Draft Standard 20543-2019, October 2019.

[AIS31] Bundesamt fuer Sicherheit in der Informationstechnik (BSI), Killmann, W., and W. Schindler, "A proposal for: Functionality classes for random number generators, version 2.0", September 2011, <https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.pdf>.

Appendix A. Reasons for the Presence of RAs

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

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

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

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

Some of the organizational reasons that argue for the presence of an RA are the following.

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

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

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

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

Further reasons relevant for automated machine-to-machine certificate lifecycle management are available in the Lightweight CMP Profile [RFCBBBB].

Appendix B. The Use of Revocation Passphrase

< ToDo: Review this Appendix and try to push the content to Section 4 or Section 5 of this document. >

< ToDo: Possibly add support for certificates containing KEM keys. >

A revocation request must incorporate suitable security mechanisms, including proper authentication, in order to reduce the probability of successful denial-of-service attacks. A digital signature on the request -- **REQUIRED** to support within this specification if revocation requests are supported -- can provide the authentication required, but there are circumstances under which an alternative mechanism may be desirable (e.g., when the private key is no longer accessible and the entity wishes to request a revocation prior to re-certification of another key pair). In order to accommodate such circumstances, a password-based MAC, see CMP Algorithms [RFCCCCC] Section 6.1, on the request is also **REQUIRED** to support within this specification (subject to local security policy for a given environment) if revocation requests are supported and if shared secret information can be established between the requester and the responder prior to the need for revocation.

A mechanism that has seen use in some environments is "revocation passphrase", in which a value of sufficient entropy (i.e., a relatively long passphrase rather than a short password) is shared between (only) the entity and the CA/RA at some point prior to revocation; this value is later used to authenticate the revocation request.

In this specification, the following technique to establish shared secret information (i.e., a revocation passphrase) is **OPTIONAL** to support. Its precise use in CMP messages is as follows.

* The OID and value specified in Section 5.3.19.9 **MAY** be sent in a GenMsg message at any time, or **MAY** be sent in the generalInfo field of the PKIHeader of any PKIMessage at any time. (In particular, the EncryptedKey structure as described in Section 5.2.2 may be sent in the header of the certConf message that confirms acceptance of certificates requested in an initialization request or certificate request message.) This conveys a revocation passphrase chosen by the entity to the relevant CA/RA. When EnvelopedData is used, this is in the decrypted bytes of encryptedContent field. When EncryptedValue is used, this is in the decrypted bytes of the encValue field. Furthermore, the transfer is accomplished with appropriate confidentiality characteristics.

* If a CA/RA receives the revocation passphrase (OID and value specified in Section 5.3.19.9) in a GenMsg, it **MUST** construct and send a GenRep message that includes the OID (with absent value) specified in Section 5.3.19.9. If the CA/RA receives the revocation passphrase in the generalInfo field of a PKIHeader of any PKIMessage, it **MUST** include the OID (with absent value) in the generalInfo field of the PKIHeader of the corresponding response PKIMessage. If the CA/RA is unable to return the appropriate response message for any reason, it **MUST** send an error message with a status of "rejection" and, optionally, a failInfo reason set.

* Either the localKeyId attribute of EnvelopedData as specified in RFC 2985 [RFC2985] or the valueHint field of EncryptedValue **MAY** contain a key identifier (chosen by the entity, along with the passphrase itself) to assist in later retrieval of the correct passphrase (e.g., when the revocation request is constructed by the entity and received by the CA/RA).

* The revocation request message is protected by a password-based MAC, see CMP Algorithms [RFCCCCC] Section 6.1, with the revocation passphrase as the key. If appropriate, the senderKID field in the PKIHeader **MAY** contain the value previously transmitted in localKeyId or valueHint.

Using the technique specified above, the revocation passphrase may be initially established and updated at any time without requiring extra messages or out-of-band exchanges. For example, the revocation request message itself (protected and authenticated through a MAC that uses the revocation passphrase as a key) may contain, in the PKIHeader, a new revocation passphrase to be used for authenticating future revocation requests for any of the entity's other certificates. In some environments this may be preferable to mechanisms that reveal the passphrase in the revocation request message, since this can allow a denial-of-service attack in which the revealed passphrase is used by an unauthorized third party to authenticate revocation requests on the entity's other certificates. However, because the passphrase is not revealed in the request message, there is no requirement that the passphrase must always be updated when a revocation request is made (that is, the same passphrase **MAY** be used by an entity to authenticate revocation requests for different certificates at different times).

Furthermore, the above technique can provide strong cryptographic protection over the entire revocation request message even when a digital signature is not used. Techniques that do authentication of the revocation request by simply revealing the revocation passphrase typically do not provide cryptographic protection over the fields of the request message (so that a request for revocation of one certificate may be modified by an unauthorized third party to a request for revocation of another certificate for that entity).

Appendix C. PKI Management Message Profiles (REQUIRED)

This appendix contains detailed profiles for those PKIMessages that **MUST** be supported by conforming implementations (see Section 6).

Note: Appendix C and D focus on PKI management operations managing certificates for human end entities. In contrast, the Lightweight CMP Profile [RFCBBBB] focuses on typical use cases of industrial and IoT scenarios supporting highly automated certificate lifecycle management scenarios.

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

* initial registration/certification

* basic authenticated scheme

* certificate request

* key update

C.1. General Rules for Interpretation of These Profiles.

1. Where **OPTIONAL** or DEFAULT fields are not mentioned in individual profiles, they **SHOULD** be absent from the relevant message (i.e., a receiver can validly reject a message containing such fields as being syntactically incorrect). Mandatory fields are not mentioned if they have an obvious value (e.g., if not explicitly stated, pvno is cmp2000(2)).

2. Where structures occur in more than one message, they are separately profiled as appropriate.

3. The algorithmIdentifiers from PKIMessage structures are profiled separately.

4. A "special" X.500 DN is called the "NULL-DN"; this means a DN containing a zero-length SEQUENCE OF RelativeDistinguishedNames (its DER encoding is then '3000'H).

5. Where a GeneralName is required for a field, but no suitable value is available (e.g., an end entity produces a request before knowing its name), then the GeneralName is to be an X.500 NULL-DN (i.e., the Name field of the CHOICE is to contain a NULL-DN). This special value can be called a "NULL-GeneralName".

6. Where a profile omits to specify the value for a GeneralName, then the NULL-GeneralName value is to be present in the relevant PKIMessage field. This occurs with the sender field of the PKIHeader for some messages.

7. Where any ambiguity arises due to naming of fields, the profile names these using a "dot" notation (e.g., "certTemplate.subject" means the subject field within a field called certTemplate).

8. Where a "SEQUENCE OF types" is part of a message, a zero-based array notation is used to describe fields within the SEQUENCE OF (e.g., crm[0].certReq.certTemplate.subject refers to a subfield of the first CertReqMsg contained in a request message).

9. All PKI message exchanges in Appendix C.4 to C.6 require a certConf message to be sent by the initiating entity and a PKIConfirm to be sent by the responding entity. The PKIConfirm is not included in some of the profiles given since its body is NULL and its header contents are clear from the context. Any authenticated means can be used for the protectionAlg (e.g., password-based MAC, if shared secret information is known, or signature).

C.2. Algorithm Use Profile

For specifications of algorithm identifiers and respective conventions for conforming implementations, please refer to CMP Algorithms Appendix 7.1 [RFCCCCC].

C.3. Proof-of-Possession Profile

POP fields for use (in signature field of pop field of ProofOfPossession structure) when proving possession of a private signing key that corresponds to a public verification key for which a certificate has been requested.

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

Note: For examples of MSG_SIG_ALG OIDs see CMP Algorithms Section 3 [RFCCCCC].

Proof-of-possession of a private decryption key that corresponds to a public encryption key for which a certificate has been requested does not use this profile; the CertHash field of the certConf message is used instead.

Not every CA/RA will do Proof-of-Possession (of signing key, decryption key, or key agreement key) in the PKIX-CMP in-band certification request protocol (how POP is done **MAY** ultimately be a policy issue that is made explicit for any given CA in its publicized Policy OID and Certification Practice Statement). However, this specification mandates that CA/RA entities **MUST** do POP (by some means) as part of the certification process. All end entities **MUST** be prepared to provide POP (i.e., these components of the PKIX-CMP protocol **MUST** be supported).

C.4. Initial Registration/Certification (Basic Authenticated Scheme)

An (uninitialized) end entity requests a (first) certificate from a CA. When the CA responds with a message containing a certificate, the end entity replies with a certificate confirmation. The CA sends a PKIConfirm back, closing the transaction. All messages are authenticated.

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

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

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

Preconditions:

1. The end entity can authenticate the CA's signature based on out-of-band means

2. The end entity and the CA share a symmetric MACing key

Message flow:

```
Step# End entity                           PKI
  1   format ir
  2                    ->   ir      ->
  3                                        handle ir
  4                                        format ip
  5                    <-   ip      <-
  6   handle ip
  7   format certConf
  8                    ->   certConf ->
  9                                        handle certConf
 10                                        format PKIConf
 11                    <-   PKIConf  <-
 12   handle PKIConf
```

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

The end entity has an out-of-band interaction with the CA/RA. This transaction established the shared secret, the referenceNumber and OPTIONALLY the distinguished name used for both sender and subject name in the certificate template. See Section 8.5 for security considerations on quality of shared secret information.

Initialization Request -- ir

```
Field                Value

recipient            CA name
  -- the name of the CA who is being asked to produce a certificate
protectionAlg        MSG_MAC_ALG
  -- only MAC protection is allowed for this request, based
  -- on initial authentication key
senderKID            referenceNum
  -- the reference number which the CA has previously issued
  -- to the end entity (together with the MACing key)
transactionID        present
  -- implementation-specific value, meaningful to end
  -- entity.
  -- [If already in use at the CA, then a rejection message MUST
  -- be produced by the CA]

senderNonce          present
  -- 128 (pseudo-)random bits
freeText             any valid value
body                 ir (CertReqMessages)
                     only one or two CertReqMsg
                     are allowed
  -- if more certificates are required, requests MUST be
  -- packaged in separate PKIMessages

CertReqMsg           one or two present
  -- see below for details, note: crm[0] means the first
  -- (which MUST be present), crm[1] means the second (which
  -- is OPTIONAL, and used to ask for a centrally-generated key)

crm[0].certReq.      fixed value of zero
   certReqId
  -- this is the index of the template within the message
crm[0].certReq       present
   certTemplate
  -- MUST include subject public key value, otherwise unconstrained
crm[0].pop...        optionally present if public key
   POPOSigningKey    from crm[0].certReq.certTemplate is
                     a signing key
  -- proof-of-possession MAY be required in this exchange
  -- (see Appendix D.3 for details)
crm[0].certReq.      optionally present
   controls.archiveOptions
  -- the end entity MAY request that the locally-generated
  -- private key be archived

crm[0].certReq.      optionally present
   controls.publicationInfo
  -- the end entity MAY ask for publication of resulting cert.
```

```
crm[1].certReq       fixed value of one
      certReqId
     -- the index of the template within the message
   crm[1].certReq       present
      certTemplate
      -- MUST NOT include actual public key bits, otherwise
      -- unconstrained (e.g., the names need not be the same as in
      -- crm[0]).  Note that subjectPublicKeyInfo MAY be present
      -- and contain an AlgorithmIdentifier followed by a
      -- zero-length BIT STRING for the subjectPublicKey if it is
      -- desired to inform the CA/RA of algorithm and parameter
      -- preferences regarding the to-be-generated key pair.

   crm[1].certReq.      present [object identifier MUST be
                                 PROT_ENC_ALG]

      controls.protocolEncrKey
     -- if centralized key generation is supported by this CA,
     -- this short-term asymmetric encryption key (generated by
     -- the end entity) will be used by the CA to encrypt (a
     -- symmetric key used to encrypt) a private key generated by
     -- the CA on behalf of the end entity

crm[1].certReq.      optionally present
   controls.archiveOptions
crm[1].certReq.      optionally present
   controls.publicationInfo
protection           present
  -- bits calculated using MSG_MAC_ALG
```

Initialization Response -- ip

```
Field                Value

sender               CA name
  -- the name of the CA who produced the message
messageTime          present
  -- time at which CA produced message
protectionAlg        MSG_MAC_ALG
  -- only MAC protection is allowed for this response
senderKID             referenceNum
  -- the reference number that the CA has previously issued to the
  -- end entity (together with the MACing key)
transactionID        present
  -- value from corresponding ir message
senderNonce          present
  -- 128 (pseudo-)random bits
recipNonce           present
  -- value from senderNonce in corresponding ir message
freeText             any valid value
body                 ip (CertRepMessage)
                     contains exactly one response
                     for each request

     -- The PKI (CA) responds to either one or two requests as
     -- appropriate.  crc[0] denotes the first (always present);
     -- crc[1] denotes the second (only present if the ir message
     -- contained two requests and if the CA supports centralized
     -- key generation).
   crc[0].              fixed value of zero
      certReqId
     -- MUST contain the response to the first request in the
     -- corresponding ir message

crc[0].status.       present, positive values allowed:
   status               "accepted", "grantedWithMods"
                     negative values allowed:
                        "rejection"
crc[0].status.       present if and only if
   failInfo          crc[0].status.status is "rejection"
crc[0].              present if and only if
   certifiedKeyPair  crc[0].status.status is
                        "accepted" or "grantedWithMods"
certificate          present unless end entity's public
                     key is an encryption key and POP
                     is done in this in-band exchange
encryptedCert        present if and only if end entity's
                     public key is an encryption key and
                     POP done in this in-band exchange
publicationInfo      optionally present
```

```
  -- indicates where certificate has been published (present
  -- at discretion of CA)

crc[1].              fixed value of one
   certReqId
  -- MUST contain the response to the second request in the
  -- corresponding ir message
crc[1].status.       present, positive values allowed:
   status               "accepted", "grantedWithMods"
                     negative values allowed:
                        "rejection"
crc[1].status.       present if and only if
   failInfo          crc[0].status.status is "rejection"
crc[1].              present if and only if
   certifiedKeyPair  crc[0].status.status is "accepted"
                     or "grantedWithMods"
certificate          present
privateKey           present
   -- Use EnvelopedData; if backward compatibility is required,
   -- use EncryptedValue, see Section 5.2.2
publicationInfo      optionally present
  -- indicates where certificate has been published (present
  -- at discretion of CA)

protection           present
  -- bits calculated using MSG_MAC_ALG
extraCerts           optionally present
  -- the CA MAY provide additional certificates to the end
  -- entity
```

Certificate confirm -- certConf

```
Field                Value

sender               present
  -- same as in ir
recipient            CA name
  -- the name of the CA who was asked to produce a certificate
transactionID        present
  -- value from corresponding ir and ip messages
senderNonce          present
  -- 128 (pseudo-) random bits
recipNonce           present
  -- value from senderNonce in corresponding ip message
protectionAlg        MSG_MAC_ALG
  -- only MAC protection is allowed for this message.  The
  -- MAC is based on the initial authentication key shared
  -- between the EE and the CA.

senderKID            referenceNum
  -- the reference number which the CA has previously issued
  -- to the end entity (together with the MACing key)

body                 certConf
  -- see Section 5.3.18, "PKI Confirmation Content", for the
  -- contents of the certConf fields.
  -- Note: two CertStatus structures are required if both an
  -- encryption and a signing certificate were sent.

protection           present
  -- bits calculated using MSG_MAC_ALG
```

Confirmation -- PKIConf

```
Field                Value

sender               present
  -- same as in ip
recipient            present
  -- sender name from certConf
transactionID        present
  -- value from certConf message
senderNonce          present
  -- 128 (pseudo-) random bits
recipNonce           present
  -- value from senderNonce from certConf message
protectionAlg        MSG_MAC_ALG
  -- only MAC protection is allowed for this message.
senderKID            referenceNum
body                 PKIConf
protection           present
  -- bits calculated using MSG_MAC_ALG
```

C.5. Certificate Request

An (initialized) end entity requests a certificate from a CA (for any reason). When the CA responds with a message containing a certificate, the end entity replies with a certificate confirmation. The CA replies with a PKIConfirm, to close the transaction. All messages are authenticated.

The profile for this exchange is identical to that given in Appendix C.4, with the following exceptions:

* sender name **SHOULD** be present

* protectionAlg of MSG_SIG_ALG **MUST** be supported (MSG_MAC_ALG **MAY** also be supported) in request, response, certConfirm, and PKIConfirm messages;

* senderKID and recipKID are only present if required for message verification;

* body is cr or cp;

* body may contain one or two CertReqMsg structures, but either CertReqMsg may be used to request certification of a locally-generated public key or a centrally-generated public key (i.e., the position-dependence requirement of Appendix C.4 is removed);

* protection bits are calculated according to the protectionAlg field.

C.6. Key Update Request

An (initialized) end entity requests a certificate from a CA (to update the key pair and/or corresponding certificate that it already possesses). When the CA responds with a message containing a certificate, the end entity replies with a certificate confirmation. The CA replies with a PKIConfirm, to close the transaction. All messages are authenticated.

The profile for this exchange is identical to that given inAppendix C.4, with the following exceptions:

1. sender name **SHOULD** be present

2. protectionAlg of MSG_SIG_ALG **MUST** be supported (MSG_MAC_ALG **MAY** also be supported) in request, response, certConfirm, and PKIConfirm messages;

3. senderKID and recipKID are only present if required for message verification;

4. body is kur or kup;

5. body may contain one or two CertReqMsg structures, but either CertReqMsg may be used to request certification of a locally-generated public key or a centrally-generated public key (i.e.,the position-dependence requirement of Appendix C.4 is removed);

6. protection bits are calculated according to the protectionAlg field;

7. regCtrl OldCertId **SHOULD** be used (unless it is clear to both sender and receiver -- by means not specified in this document -- that it is not needed).

Appendix D. PKI Management Message Profiles (OPTIONAL)

This appendix contains detailed profiles for those PKIMessages that **MAY** be supported by implementations.

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

*root CA key update

*information request/response

*cross-certification request/response (1-way)

* in-band initialization using external identity certificate

Later versions of this document may extend the above to include profiles for the operations listed below (along with other operations, if desired).

* revocation request

* certificate publication

* CRL publication

D.1. General Rules for Interpretation of These Profiles.

Identical to Appendix C.1.

D.2. Algorithm Use Profile

Identical to Appendix C.2.

D.3. Self-Signed Certificates

Profile of how a Certificate structure may be "self-signed". These structures are used for distribution of CA public keys. This can occur in one of three ways (see Section 4.4 above for a description of the use of these structures):

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

Such certificates (including relevant extensions) must contain "sensible" values for all fields. For example, when present, subjectAltName **MUST** be identical to issuerAltName, and, when present, keyIdentifiers must contain appropriate values, et cetera.

D.4. Root CA Key Update

A root CA updates its key pair. It then produces a CA key update announcement message that can be made available (via some transport mechanism) to the relevant end entities. A confirmation message is not required from the end entities.

ckuann message:

Field	Value	Comment
sender	CA name CA name	
body	ckuann(CAKeyUpdAnnContent)	
oldWithNew	present	see Appendix E.3 above
newWithOld	present	see Appendix E.3 above
newWithNew	present	see Appendix E.3 above
extraCerts	optionally present	can be used to "publish" certificates (e.g., certificates signed using the new private key)

D.5. PKI Information Request/Response

The end entity sends a general message to the PKI requesting details that will be required for later PKI management operations. RA/CA responds with a general response. If an RA generates the response, then it will simply forward the equivalent message that it previously received from the CA, with the possible addition of certificates to the extraCerts fields of the PKIMessage. A confirmation message is not required from the end entity.

Message Flows:

```
Step# End entity                           PKI

   1  format genm
   2               ->   genm   ->
   3                                       handle genm
   4                                       produce genp
   5               <-   genp   <-
   6  handle genp
```

genM:

```
Field               Value

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

genP:

```
Field                 Value

sender                CA name
  -- name of the CA which produced the message
protectionAlg         MSG_MAC_ALG or MSG_SIG_ALG
  -- any authenticated protection alg.
senderKID             present if required
  -- must be present if required for verification of message
  -- protection
body                  genp (GenRepContent)
CAProtEncCert         present (object identifier one
                      of PROT_ENC_ALG), with relevant
                      value
  -- to be used if end entity needs to encrypt information for
  -- the CA (e.g., private key for recovery purposes)

SignKeyPairTypes      present, with relevant value
  -- the set of signature algorithm identifiers that this CA will
  -- certify for subject public keys
EncKeyPairTypes       present, with relevant value
  -- the set of encryption/key agreement algorithm identifiers that
  -- this CA will certify for subject public keys
PreferredSymmAlg      present (object identifier one
                      of PROT_SYM_ALG) , with relevant
                      value
  -- the symmetric algorithm that this CA expects to be used
  -- in later PKI messages (for encryption)
CAKeyUpdateInfo       optionally present, with
                      relevant value
  -- the CA MAY provide information about a relevant root CA
  -- key pair using this field (note that this does not imply
  -- that the responding CA is the root CA in question)
CurrentCRL            optionally present, with relevant value
  -- the CA MAY provide a copy of a complete CRL (i.e.,
  -- fullest possible one)
protection            present
  -- bits calculated using MSG_MAC_ALG or MSG_SIG_ALG
extraCerts            optionally present
  -- can be used to send some certificates to the end
  -- entity. An RA MAY add its certificate here.
```

D.6. Cross Certification Request/Response (1-way)

Creation of a single cross-certificate (i.e., not two at once). The requesting CA **MAY** choose who is responsible for publication of the cross-certificate created by the responding CA through use of the PKIPublicationInfo control.

Preconditions:

1. Responding CA can verify the origin of the request (possibly requiring out-of-band means) before processing the request.

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

The use of certificate confirmation and the corresponding server confirmation is determined by the generalInfo field in the PKIHeader (see Section 5.1.1). The following profile does not mandate support for either confirmation.

Message Flows:

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

ccr:

```
Field                 Value

sender                Requesting CA name
  -- the name of the CA who produced the message
recipient             Responding CA name
  -- the name of the CA who is being asked to produce a certificate
messageTime           time of production of message
  -- current time at requesting CA
protectionAlg         MSG_SIG_ALG
  -- only signature protection is allowed for this request
senderKID             present if required
  -- must be present if required for verification of message
  -- protection
recipKID             present if required
  -- must be present if required for verification of message
  -- protection
transactionID         present
  -- implementation-specific value, meaningful to requesting CA.
  -- [If already in use at responding CA then a rejection message
  -- MUST be produced by responding CA]
senderNonce           present
  -- 128 (pseudo-)random bits
freeText              any valid value
body                  ccr (CertReqMessages)
                      only one CertReqMsg
                      allowed
  -- if multiple cross certificates are required, they MUST be
  -- packaged in separate PKIMessages
certTemplate          present
  -- details follow
version               v1 or v3
  -- v3 STRONGLY RECOMMENDED
signingAlg            present
  -- the requesting CA must know in advance with which algorithm it
  -- wishes the certificate to be signed

subject               present
  -- may be NULL-DN only if subjectAltNames extension value proposed
validity              present
  -- MUST be completely specified (i.e., both fields present)
issuer                present
  -- may be NULL-DN only if issuerAltNames extension value proposed
publicKey             present
  -- the key to be certified (which must be for a signing algorithm)
extensions            optionally present
  -- a requesting CA must propose values for all extensions
  -- that it requires to be in the cross-certificate
POPOSigningKey        present
  -- see Section D3: Proof-of-possession profile
```

```
protection            present
  -- bits calculated using MSG_SIG_ALG
extraCerts            optionally present
  -- MAY contain any additional certificates that requester wishes
  -- to include
```

ccp:

```
Field                 Value

sender                Responding CA name
  -- the name of the CA who produced the message
recipient             Requesting CA name
  -- the name of the CA who asked for production of a certificate
messageTime           time of production of message
  -- current time at responding CA
protectionAlg         MSG_SIG_ALG
  -- only signature protection is allowed for this message
senderKID             present if required
  -- must be present if required for verification of message
  -- protection
recipKID              present if required
transactionID         present
  -- value from corresponding ccr message
senderNonce           present
  -- 128 (pseudo-)random bits
recipNonce            present
-- senderNonce from corresponding ccr message
freeText              any valid value
body                  ccp (CertRepMessage)
                      only one CertResponse allowed
  -- if multiple cross certificates are required they MUST be
  -- packaged in separate PKIMessages
response              present
status                present

PKIStatusInfo.status  present
  -- if PKIStatusInfo.status is one of:
  --   accepted, or
  --   grantedWithMods,
  -- then certifiedKeyPair MUST be present and failInfo MUST
  -- be absent

failInfo              present depending on
                      PKIStatusInfo.status
  -- if PKIStatusInfo.status is:
  --   rejection
  -- then certifiedKeyPair MUST be absent and failInfo MUST be
  -- present and contain appropriate bit settings

certifiedKeyPair      present depending on
                      PKIStatusInfo.status
certificate           present depending on
                      certifiedKeyPair
  -- content of actual certificate must be examined by requesting CA
  -- before publication
protection            present
```

```
  -- bits calculated using MSG_SIG_ALG
extraCerts            optionally present
  -- MAY contain any additional certificates that responder wishes
  -- to include
```

D.7. In-Band Initialization Using External Identity Certificate

An (uninitialized) end entity wishes to initialize into the PKI with a CA, CA-1. It uses, for authentication purposes, a pre-existing identity certificate issued by another (external) CA, CA-X. A trust relationship must already have been established between CA-1 and CA-X so that CA-1 can validate the EE identity certificate signed by CA-X. Furthermore, some mechanism must already have been established within the Personal Security Environment (PSE) of the EE that would allow it to authenticate and verify PKIMessages signed by CA-1 (as one example, the PSE may contain a certificate issued for the public key of CA-1, signed by another CA that the EE trusts on the basis of out-of-band authentication techniques).

The EE sends an initialization request to start the transaction. When CA-1 responds with a message containing the new certificate, the end entity replies with a certificate confirmation. CA-1 replies with a PKIConfirm to close the transaction. All messages are signed (the EE messages are signed using the private key that corresponds to the public key in its external identity certificate; the CA-1 messages are signed using the private key that corresponds to the public key in a

certificate that can be chained to a trust anchor in the EE's PSE).

The profile for this exchange is identical to that given in Appendix C.4, with the following exceptions:

* the EE and CA-1 do not share a symmetric MACing key (i.e., there is no out-of-band shared secret information between these entities);

* sender name in ir **MUST** be present (and identical to the subject name present in the external identity certificate);

* protectionAlg of MSG_SIG_ALG **MUST** be used in all messages;

* external identity cert. **MUST** be carried in ir extraCerts field

* senderKID and recipKID are not used;

* body is ir or ip;

* protection bits are calculated according to the protectionAlg field.

Appendix E. Compilable ASN.1 Definitions

This section contains the updated 2002 ASN.1 module for [RFC5912] as updated in [RFCAAAA]. This module replaces the module in Section 9 of [RFC5912]. The module contains those changes to the normative ASN.1 module from RFC 4210 Appendix F [RFC4210] that were specified in [RFCAAAA] as well as changes made in this document.

```
PKIXCMP-2021
    { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-cmp2021-02(100) }
DEFINITIONS EXPLICIT TAGS ::=
BEGIN
IMPORTS

AttributeSet{}, SingleAttribute{}, Extensions{}, 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{}, SIGNATURE-ALGORITHM, ALGORITHM,
    DIGEST-ALGORITHM, MAC-ALGORITHM
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)}

Certificate, CertificateList, Time, 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)}

DistributionPointName, GeneralNames, GeneralName, KeyIdentifier
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)}

CertTemplate, PKIPublicationInfo, EncryptedKey, CertId,
    CertReqMessages, Controls, RegControlSet, id-regCtrl
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) }
    -- The import of EncryptedKey is added due to the updates made
    -- in CMP Updates [RFCAAAA]. EncryptedValue does not need to
    -- be imported anymore and is therefore removed here.

CertificationRequest
FROM PKCS-10
    {iso(1) identified-organization(3) dod(6) internet(1) security(5)
    mechanisms(5) pkix(7) id-mod(0) id-mod-pkcs10-2009(69)}
-- (specified in RFC 2986 with 1993 ASN.1 syntax and IMPLICIT
-- tags).  Alternatively, implementers may directly include
-- the [RFC2986] syntax in this module

localKeyId
```

```
FROM PKCS-9
    {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
    modules(0) pkcs-9(1)}
    -- The import of localKeyId is added due to the updates made in
    -- CMP Updates [RFCAAAA]

EnvelopedData, SignedData
FROM CryptographicMessageSyntax-2009
    {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
    smime(16) modules(0) id-mod-cms-2004-02(41)}
    -- The import of EnvelopedData and SignedData is added due to
    -- the updates made in CMP Updates [RFCAAAA]
;

-- the rest of the module contains locally defined OIDs and
-- constructs

CMPCertificate ::= CHOICE { x509v3PKCert Certificate, ... }
-- This syntax, while bits-on-the-wire compatible with the
-- standard X.509 definition of "Certificate", allows the
-- possibility of future certificate types (such as X.509
-- attribute certificates, WAP WTLS certificates, or other kinds
-- of certificates) within this certificate management protocol,
-- should a need ever arise to support such generality.  Those
-- implementations that do not foresee a need to ever support
-- other certificate types MAY, if they wish, comment out the
-- above structure and "uncomment" the following one prior to
-- compiling this ASN.1 module.  (Note that interoperability
-- with implementations that don't do this will be unaffected by
-- this change.)

-- CMPCertificate ::= Certificate

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

PKIMessages ::= SEQUENCE SIZE (1..MAX) OF PKIMessage

PKIHeader ::= SEQUENCE {
    pvno                INTEGER     { cmp1999(1), cmp2000(2),
                                      cmp2012(3) },
    sender              GeneralName,
    -- identifies the sender
    recipient           GeneralName,
    -- identifies the intended recipient
```

```
    messageTime     [0] GeneralizedTime         OPTIONAL,
    -- time of production of this message (used when sender
    -- believes that the transport will be "suitable"; i.e.,
    -- that the time will still be meaningful upon receipt)
    protectionAlg   [1] AlgorithmIdentifier{ALGORITHM, {...}}
                            OPTIONAL,
    -- algorithm used for calculation of protection bits
    senderKID       [2] KeyIdentifier           OPTIONAL,
    recipKID        [3] KeyIdentifier           OPTIONAL,
    -- to identify specific keys used for protection
    transactionID   [4] OCTET STRING            OPTIONAL,
    -- identifies the transaction; i.e., this will be the same in
    -- corresponding request, response, certConf, and PKIConf
    -- messages
    senderNonce     [5] OCTET STRING            OPTIONAL,
    recipNonce      [6] OCTET STRING            OPTIONAL,
    -- nonces used to provide replay protection, senderNonce
    -- is inserted by the creator of this message; recipNonce
    -- is a nonce previously inserted in a related message by
    -- the intended recipient of this message
    freeText        [7] PKIFreeText             OPTIONAL,
    -- this may be used to indicate context-specific instructions
    -- (this field is intended for human consumption)
    generalInfo     [8] SEQUENCE SIZE (1..MAX) OF
                        InfoTypeAndValue     OPTIONAL
    -- this may be used to convey context-specific information
    -- (this field not primarily intended for human consumption)
}

PKIFreeText ::= SEQUENCE SIZE (1..MAX) OF UTF8String
    -- text encoded as UTF-8 String [RFC3629]

PKIBody ::= CHOICE {       -- message-specific body elements
    ir       [0]  CertReqMessages,        --Initialization Request
    ip       [1]  CertRepMessage,         --Initialization Response
    cr       [2]  CertReqMessages,        --Certification Request
    cp       [3]  CertRepMessage,         --Certification Response
    p10cr    [4]  CertificationRequest,   --imported from [RFC2986]
    popdecc  [5]  POPODecKeyChallContent, --pop Challenge
    popdecr  [6]  POPODecKeyRespContent,  --pop Response
    kur      [7]  CertReqMessages,        --Key Update Request
    kup      [8]  CertRepMessage,         --Key Update Response
    krr      [9]  CertReqMessages,        --Key Recovery Request
    krp      [10] KeyRecRepContent,       --Key Recovery Response
    rr       [11] RevReqContent,          --Revocation Request
    rp       [12] RevRepContent,          --Revocation Response
    ccr      [13] CertReqMessages,        --Cross-Cert. Request
    ccp      [14] CertRepMessage,         --Cross-Cert. Response
    ckuann   [15] CAKeyUpdAnnContent,     --CA Key Update Ann.
```

```
    cann     [16] CertAnnContent,         --Certificate Ann.
    rann     [17] RevAnnContent,          --Revocation Ann.
    crlann   [18] CRLAnnContent,          --CRL Announcement
    pkiconf  [19] PKIConfirmContent,      --Confirmation
    nested   [20] NestedMessageContent,   --Nested Message
    genm     [21] GenMsgContent,          --General Message
    genp     [22] GenRepContent,          --General Response
    error    [23] ErrorMsgContent,        --Error Message
    certConf [24] CertConfirmContent,     --Certificate confirm
    pollReq  [25] PollReqContent,         --Polling request
    pollRep  [26] PollRepContent          --Polling response
}

PKIProtection ::= BIT STRING

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

id-PasswordBasedMac OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    usa(840) nt(113533) nsn(7) algorithms(66) 13 }
PBMParameter ::= SEQUENCE {
    salt                OCTET STRING,
    -- note:  implementations MAY wish to limit acceptable sizes
    -- of this string to values appropriate for their environment
    -- in order to reduce the risk of denial-of-service attacks
    owf                 AlgorithmIdentifier{DIGEST-ALGORITHM, {...}},
    -- AlgId for a One-Way Function
    iterationCount      INTEGER,
    -- number of times the OWF is applied
    -- note:  implementations MAY wish to limit acceptable sizes
    -- of this integer to values appropriate for their environment
    -- in order to reduce the risk of denial-of-service attacks
    mac                 AlgorithmIdentifier{MAC-ALGORITHM, {...}}
    -- the MAC AlgId
}

id-DHBasedMac OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    usa(840) nt(113533) nsn(7) algorithms(66) 30 }
DHBMParameter ::= SEQUENCE {
    owf                 AlgorithmIdentifier{DIGEST-ALGORITHM, {...}},
    -- AlgId for a One-Way Function
    mac                 AlgorithmIdentifier{MAC-ALGORITHM, {...}}
    -- the MAC AlgId
}

PKIStatus ::= INTEGER {
    accepted               (0),
    -- you got exactly what you asked for
```

```
    grantedWithMods        (1),
    -- you got something like what you asked for; the
    -- requester is responsible for ascertaining the differences
    rejection              (2),
    -- you don't get it, more information elsewhere in the message
    waiting                (3),
    -- the request body part has not yet been processed; expect to
    -- hear more later (note: proper handling of this status
    -- response MAY use the polling req/rep PKIMessages specified
    -- in Section 5.3.22; alternatively, polling in the underlying
    -- transport layer MAY have some utility in this regard)
    revocationWarning      (4),
    -- this message contains a warning that a revocation is
    -- imminent
    revocationNotification (5),
    -- notification that a revocation has occurred
    keyUpdateWarning       (6)
    -- update already done for the oldCertId specified in
    -- CertReqMsg
}

PKIFailureInfo ::= BIT STRING {
-- since we can fail in more than one way!
-- More codes may be added in the future if/when required.
    badAlg              (0),
    -- unrecognized or unsupported Algorithm Identifier
    badMessageCheck     (1),
    -- integrity check failed (e.g., signature did not verify)
    badRequest          (2),
    -- transaction not permitted or supported
    badTime             (3),
    -- messageTime was not sufficiently close to the system time,
    -- as defined by local policy
    badCertId           (4),
    -- no certificate could be found matching the provided criteria
    badDataFormat       (5),
    -- the data submitted has the wrong format
    wrongAuthority      (6),
    -- the authority indicated in the request is different from the
    -- one creating the response token
    incorrectData       (7),
    -- the requester's data is incorrect (for notary services)
    missingTimeStamp    (8),
    -- when the timestamp is missing but should be there
    -- (by policy)
    badPOP              (9),
    -- the proof-of-possession failed
    certRevoked         (10),
    -- the certificate has already been revoked
```

```
    certConfirmed       (11),
    -- the certificate has already been confirmed
    wrongIntegrity      (12),
    -- not valid integrity, password based instead of signature or
    -- vice versa
    badRecipientNonce   (13),
    -- not valid recipient nonce, either missing or wrong value
    timeNotAvailable    (14),
    -- the TSA's time source is not available
    unacceptedPolicy    (15),
    -- the requested TSA policy is not supported by the TSA
    unacceptedExtension (16),
    -- the requested extension is not supported by the TSA
    addInfoNotAvailable (17),
    -- the additional information requested could not be
    -- understood or is not available
    badSenderNonce      (18),
    -- not valid sender nonce, either missing or wrong size
    badCertTemplate     (19),
    -- not valid cert. template or missing mandatory information
    signerNotTrusted    (20),
    -- signer of the message unknown or not trusted
    transactionIdInUse  (21),
    -- the transaction identifier is already in use
    unsupportedVersion  (22),
    -- the version of the message is not supported
    notAuthorized       (23),
    -- the sender was not authorized to make the preceding
    -- request or perform the preceding action
    systemUnavail       (24),
    -- the request cannot be handled due to system unavailability
    systemFailure       (25),
    -- the request cannot be handled due to system failure
    duplicateCertReq    (26)
    -- certificate cannot be issued because a duplicate
    -- certificate already exists
}

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

OOBCert ::= CMPCertificate

OOBCertHash ::= SEQUENCE {
    hashAlg     [0] AlgorithmIdentifier{DIGEST-ALGORITHM, {...}}
                        OPTIONAL,
    certId      [1] CertId                  OPTIONAL,
```

```
    hashVal         BIT STRING
    -- hashVal is calculated over the DER encoding of the
    -- self-signed certificate with the identifier certID.
}

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

Challenge ::= SEQUENCE {
    owf                 AlgorithmIdentifier{DIGEST-ALGORITHM, {...}}
                            OPTIONAL,
    -- MUST be present in the first Challenge; MAY be omitted in
    -- any subsequent Challenge in POPODecKeyChallContent (if
    -- omitted, then the owf used in the immediately preceding
    -- Challenge is to be used).
    witness             OCTET STRING,
    -- the result of applying the one-way function (owf) to a
    -- randomly-generated INTEGER, A.  [Note that a different
    -- INTEGER MUST be used for each Challenge.]
    challenge           OCTET STRING
    -- the encryption (under the public key for which the cert.
    -- request is being made) of Rand.
}

-- Added in CMP Updates [RFCAAAA]

Rand ::= SEQUENCE {
-- Rand is encrypted under the public key to form the challenge
-- in POPODecKeyChallContent
   int                  INTEGER,
   -- the randomly-generated INTEGER A (above)
   sender               GeneralName
   -- the sender's name (as included in PKIHeader)
}

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

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

CertResponse ::= SEQUENCE {
    certReqId           INTEGER,
```

```
    -- to match this response with the corresponding request (a value
    -- of -1 is to be used if certReqId is not specified in the
    -- corresponding request, which can only be a p10cr)
    status              PKIStatusInfo,
    certifiedKeyPair    CertifiedKeyPair    OPTIONAL,
    rspInfo             OCTET STRING        OPTIONAL
    -- analogous to the id-regInfo-utf8Pairs string defined
    -- for regInfo in CertReqMsg [RFC4211]
}

CertifiedKeyPair ::= SEQUENCE {
    certOrEncCert       CertOrEncCert,
    privateKey      [0] EncryptedKey      OPTIONAL,
    -- see [RFC4211] for comment on encoding
    -- Changed from Encrypted Value to EncryptedKey as a CHOICE of
    -- EncryptedValue and EnvelopedData due to the changes made in
    -- CMP Updates [RFCAAAA]
    -- Using the choice EncryptedValue is bit-compatible to the
    -- syntax without this change
    publicationInfo [1] PKIPublicationInfo  OPTIONAL }

CertOrEncCert ::= CHOICE {
    certificate     [0] CMPCertificate,
    encryptedCert   [1] EncryptedKey
    -- Changed from Encrypted Value to EncryptedKey as a CHOICE of
    -- EncryptedValue and EnvelopedData due to the changes made in
    -- CMP Updates [RFCAAAA]
    -- Using the choice EncryptedValue is bit-compatible to the
    -- syntax without this change
}

KeyRecRepContent ::= SEQUENCE {
    status                  PKIStatusInfo,
    newSigCert          [0] CMPCertificate OPTIONAL,
    caCerts             [1] SEQUENCE SIZE (1..MAX) OF
                                     CMPCertificate OPTIONAL,
    keyPairHist         [2] SEQUENCE SIZE (1..MAX) OF
                                     CertifiedKeyPair OPTIONAL }

RevReqContent ::= SEQUENCE OF RevDetails

RevDetails ::= SEQUENCE {
    certDetails         CertTemplate,
    -- allows requester to specify as much as they can about
    -- the cert. for which revocation is requested
    -- (e.g., for cases in which serialNumber is not available)
    crlEntryDetails     Extensions{{...}}    OPTIONAL
    -- requested crlEntryExtensions
}
```

```
RevRepContent ::= SEQUENCE {
    status       SEQUENCE SIZE (1..MAX) OF PKIStatusInfo,
    -- in same order as was sent in RevReqContent
    revCerts [0] SEQUENCE SIZE (1..MAX) OF CertId OPTIONAL,
    -- IDs for which revocation was requested
    -- (same order as status)
    crls     [1] SEQUENCE SIZE (1..MAX) OF CertificateList OPTIONAL
    -- the resulting CRLs (there may be more than one)
}

CAKeyUpdAnnContent ::= SEQUENCE {
    oldWithNew   CMPCertificate, -- old pub signed with new priv
    newWithOld   CMPCertificate, -- new pub signed with old priv
    newWithNew   CMPCertificate  -- new pub signed with new priv
}

CertAnnContent ::= CMPCertificate

RevAnnContent ::= SEQUENCE {
    status              PKIStatus,
    certId              CertId,
    willBeRevokedAt     GeneralizedTime,
    badSinceDate        GeneralizedTime,
    crlDetails          Extensions{{...}}  OPTIONAL
    -- extra CRL details (e.g., crl number, reason, location, etc.)
}

CRLAnnContent ::= SEQUENCE OF CertificateList
PKIConfirmContent ::= NULL

NestedMessageContent ::= PKIMessages

-- CertReqTemplateContent, AttributeTypeAndValue,
-- ExpandedRegControlSet, id-regCtrl-altCertTemplate,
-- AltCertTemplate, regCtrl-algId, id-regCtrl-algId, AlgIdCtrl,
-- regCtrl-rsaKeyLen, id-regCtrl-rsaKeyLen, and RsaKeyLenCtrl
-- were added in CMP Updates [RFCAAAA]

CertReqTemplateContent ::= SEQUENCE {
   certTemplate           CertTemplate,
   -- prefilled certTemplate structure elements
   -- The SubjectPublicKeyInfo field in the certTemplate MUST NOT
   -- be used.
   keySpec                Controls OPTIONAL
   -- MAY be used to specify supported algorithms.
   -- Controls  ::= SEQUENCE SIZE (1..MAX) OF AttributeTypeAndValue
   -- as specified in CRMF (RFC4211)
   }
```

```
AttributeTypeAndValue ::= SingleAttribute{{ ... }}

ExpandedRegControlSet ATTRIBUTE ::= { RegControlSet |
   regCtrl-altCertTemplate | regCtrl-algId | regCtrl-rsaKeyLen, ... }

regCtrl-altCertTemplate ATTRIBUTE ::=
   { TYPE AltCertTemplate IDENTIFIED BY id-regCtrl-altCertTemplate }

id-regCtrl-altCertTemplate OBJECT IDENTIFIER ::= { id-regCtrl 7 }

AltCertTemplate ::= AttributeTypeAndValue
   -- specifies a template for a certificate other than an X.509v3
   -- public-key certificate

regCtrl-algId ATTRIBUTE ::=
   { TYPE AlgIdCtrl IDENTIFIED BY id-regCtrl-algId }

id-regCtrl-algId OBJECT IDENTIFIER ::= { id-regCtrl 11 }

AlgIdCtrl ::= AlgorithmIdentifier{ALGORITHM, {...}}
   -- SHALL be used to specify supported algorithms other than RSA

regCtrl-rsaKeyLen ATTRIBUTE ::=
   { TYPE RsaKeyLenCtrl IDENTIFIED BY id-regCtrl-rsaKeyLen }

id-regCtrl-rsaKeyLen OBJECT IDENTIFIER ::= { id-regCtrl 12 }

RsaKeyLenCtrl ::= INTEGER (1..MAX)
   -- SHALL be used to specify supported RSA key lengths

-- RootCaKeyUpdateContent, CRLSource, and CRLStatus were added in
-- CMP Updates [RFCAAAA]

RootCaKeyUpdateContent ::= SEQUENCE {
   newWithNew       CMPCertificate,
   -- new root CA certificate
   newWithOld   [0] CMPCertificate OPTIONAL,
   -- X.509 certificate containing the new public root CA key
   -- signed with the old private root CA key
   oldWithNew   [1] CMPCertificate OPTIONAL
   -- X.509 certificate containing the old public root CA key
   -- signed with the new private root CA key
   }

CRLSource ::= CHOICE {
   dpn          [0] DistributionPointName,
   issuer       [1] GeneralNames }

CRLStatus ::= SEQUENCE {
   source       CRLSource,
```

```
   thisUpdate   Time OPTIONAL }

INFO-TYPE-AND-VALUE ::= TYPE-IDENTIFIER

InfoTypeAndValue ::= SEQUENCE {
    infoType    INFO-TYPE-AND-VALUE.
                    &id({SupportedInfoSet}),
    infoValue   INFO-TYPE-AND-VALUE.
                    &Type({SupportedInfoSet}{@infoType}) }

SupportedInfoSet INFO-TYPE-AND-VALUE ::= { ... }

-- Example InfoTypeAndValue contents include, but are not limited
-- to, the following (uncomment in this ASN.1 module and use as
-- appropriate for a given environment):
--
--   id-it-caProtEncCert    OBJECT IDENTIFIER ::= {id-it 1}
--      CAProtEncCertValue      ::= CMPCertificate
--   id-it-signKeyPairTypes OBJECT IDENTIFIER ::= {id-it 2}
--      SignKeyPairTypesValue   ::= SEQUENCE SIZE (1..MAX) OF
--                                      AlgorithmIdentifier{{...}}
--   id-it-encKeyPairTypes  OBJECT IDENTIFIER ::= {id-it 3}
--      EncKeyPairTypesValue    ::= SEQUENCE SIZE (1..MAX) OF
--                                      AlgorithmIdentifier{{...}}
--   id-it-preferredSymmAlg OBJECT IDENTIFIER ::= {id-it 4}
--      PreferredSymmAlgValue   ::= AlgorithmIdentifier{{...}}
--   id-it-caKeyUpdateInfo  OBJECT IDENTIFIER ::= {id-it 5}
--      CAKeyUpdateInfoValue    ::= CAKeyUpdAnnContent
--   id-it-currentCRL       OBJECT IDENTIFIER ::= {id-it 6}
--      CurrentCRLValue         ::= CertificateList
--   id-it-unsupportedOIDs  OBJECT IDENTIFIER ::= {id-it 7}
--      UnsupportedOIDsValue    ::= SEQUENCE SIZE (1..MAX) OF
--                                          OBJECT IDENTIFIER
--   id-it-keyPairParamReq  OBJECT IDENTIFIER ::= {id-it 10}
--      KeyPairParamReqValue    ::= OBJECT IDENTIFIER
--   id-it-keyPairParamRep  OBJECT IDENTIFIER ::= {id-it 11}
--      KeyPairParamRepValue    ::= AlgorithmIdentifier{{...}}
--   id-it-revPassphrase    OBJECT IDENTIFIER ::= {id-it 12}
--      RevPassphraseValue      ::= EncryptedKey
--      - Changed from Encrypted Value to EncryptedKey as a CHOICE
--      - of EncryptedValue and EnvelopedData due to the changes
--      - made in CMP Updates [RFCAAAA]
--      - Using the choice EncryptedValue is bit-compatible to
--      - the syntax without this change
--   id-it-implicitConfirm  OBJECT IDENTIFIER ::= {id-it 13}
--      ImplicitConfirmValue    ::= NULL
--   id-it-confirmWaitTime  OBJECT IDENTIFIER ::= {id-it 14}
--      ConfirmWaitTimeValue    ::= GeneralizedTime
--   id-it-origPKIMessage   OBJECT IDENTIFIER ::= {id-it 15}
```

```
--      OrigPKIMessageValue     ::= PKIMessages
--   id-it-suppLangTags     OBJECT IDENTIFIER ::= {id-it 16}
--      SuppLangTagsValue       ::= SEQUENCE OF UTF8String
--   id-it-caCerts          OBJECT IDENTIFIER ::= {id-it 17}
--      CaCertsValue            ::= SEQUENCE SIZE (1..MAX) OF
--                                              CMPCertificate
--      - id-it-caCerts added in CMP Updates [RFCAAAA]
--   id-it-rootCaKeyUpdate  OBJECT IDENTIFIER ::= {id-it 18}
--      RootCaKeyUpdateValue    ::= RootCaKeyUpdateContent
--      - id-it-rootCaKeyUpdate added in CMP Updates [RFCAAAA]
--   id-it-certReqTemplate  OBJECT IDENTIFIER ::= {id-it 19}
--      CertReqTemplateValue    ::= CertReqTemplateContent
--      - id-it-certReqTemplate added in CMP Updates [RFCAAAA]
--   id-it-rootCaCert       OBJECT IDENTIFIER ::= {id-it 20}
--      RootCaCertValue         ::= CMPCertificate
--      - id-it-rootCaCert added in CMP Updates [RFCAAAA]
--   id-it-certProfile      OBJECT IDENTIFIER ::= {id-it 21}
--      CertProfileValue        ::= SEQUENCE SIZE (1..MAX) OF
--                                                  UTF8String
--      - id-it-certProfile added in CMP Updates [RFCAAAA]
--   id-it-crlStatusList    OBJECT IDENTIFIER ::= {id-it 22}
--   CRLStatusListValue         ::= SEQUENCE SIZE (1..MAX) OF
--                                                   CRLStatus
--      - id-it-crlStatusList added in CMP Updates [RFCAAAA]
--   id-it-crls             OBJECT IDENTIFIER ::= {id-it 23}
--   CRLsValue                  ::= SEQUENCE SIZE (1..MAX) OF
--                                             CertificateList
--      - id-it-crls added in CMP Updates [RFCAAAA]
--
-- where
--
--   id-pkix OBJECT IDENTIFIER ::= {
--      iso(1) identified-organization(3)
--      dod(6) internet(1) security(5) mechanisms(5) pkix(7)}
-- and
--   id-it   OBJECT IDENTIFIER ::= {id-pkix 4}
--
--
-- This construct MAY also be used to define new PKIX Certificate
-- Management Protocol request and response messages, or
-- general-purpose (e.g., announcement) messages for future needs
-- or for specific environments.

GenMsgContent ::= SEQUENCE OF InfoTypeAndValue

-- May be sent by EE, RA, or CA (depending on message content).
-- The OPTIONAL infoValue parameter of InfoTypeAndValue will
-- typically be omitted for some of the examples given above.
-- The receiver is free to ignore any contained OBJECT IDs that it
```

```
-- does not recognize.  If sent from EE to CA, the empty set
-- indicates that the CA may send
-- any/all information that it wishes.

GenRepContent ::= SEQUENCE OF InfoTypeAndValue
-- Receiver MAY ignore any contained OIDs that it does not
-- recognize.

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

CertConfirmContent ::= SEQUENCE OF CertStatus

CertStatus ::= SEQUENCE {
    certHash    OCTET STRING,
    -- the hash of the certificate, using the same hash algorithm
    -- as is used to create and verify the certificate signature
    certReqId   INTEGER,
    -- to match this confirmation with the corresponding req/rep
    statusInfo  PKIStatusInfo OPTIONAL,
    hashAlg [0] AlgorithmIdentifier{DIGEST-ALGORITHM, {...}} OPTIONAL
    -- the hash algorithm to use for calculating certHash
    -- SHOULD NOT be used in all cases where the AlgorithmIdentifier
    -- of the certificate signature specifies a hash algorithm
   }

PollReqContent ::= SEQUENCE OF SEQUENCE {
    certReqId              INTEGER }

PollRepContent ::= SEQUENCE OF SEQUENCE {
    certReqId              INTEGER,
    checkAfter             INTEGER,  -- time in seconds
    reason                 PKIFreeText OPTIONAL }

--
-- Extended Key Usage extension for PKI entities used in CMP
-- operations, added due to the changes made in
-- CMP Updates [RFCAAAA]
-- The EKUs for the CA and RA are reused from CMC as defined in
-- [RFC6402]
--

-- id-kp-cmcCA OBJECT IDENTIFIER ::= { id-kp 27 }
-- id-kp-cmcRA OBJECT IDENTIFIER ::= { id-kp 28 }
id-kp-cmKGA OBJECT IDENTIFIER ::= { id-kp 32 }
```

END

Appendix F. History of Changes

Note: This appendix will be deleted in the final version of the document.

From version 05 -> 06:

* Updated section 5.1.3.4 exchanging HPKE with plain KEM+KDF as also used in draft-ietf-lamps-cms-kemri

From version 04 -> 05:

* Updated sections 5.1.3.4, 5.2.2, and 8.9 addressing comments from Russ (see thread "I-D Action: draft-ietf-lamps-rfc4210bis-04.txt")

From version 03 -> 04:

* Added Section 4.3.4 regarding POP for KEM keys

* Added Section 5.1.3.4 on message protection using KEM keys and HPKE

* Aligned Section 5.2.2 on guidance which CMS key management technique to use with encrypted values (see thread "CMS: selection of key management technique to use for EnvelopedData") also adding support for KEM keys

* Added Section 8.9 and extended Section 3.1.2 regarding use of Certificate Transparency logs

* Deleted former Appendix C as announced in the -03

* Fixed some nits resulting from XML -> MD conversion

From version 02 -> 03:

* Updated Section 4.4.1 clarifying the definition of "new with new" certificate validity period (see thread "RFC4210bis - notAfter time of newWithNew certificate")

* Added ToDo to Section 4.3 and 5.2.8 on required alignment regarding POP for KEM keys.

* Updated Sections 5.2.1, 5.2.8, and 5.2.8.1 incorporating text of former Appendix C (see thread "draft-ietf-lamps-rfc4210bis - ToDo on review of Appendix C")

* Added a ToDo to Appendix B to indicate additional review need to try pushing the content to Sections 4 and Section 5

From version 01 -> 02:

* Added Section 3.1.1.4 introducing the Key Generation Authority
* Added Section 5.1.1.3 containing description of origPKIMessage content moved here from Section 5.1.3.4
* Added ToDos on defining POP and message protection using KEM keys
* Added a ToDo to Section 4.4.3
* Added a ToDo to Appendix C to do a more detailed review
* Removed concrete algorithms and referred to CMP Algorithms instead
* Added references to Appendix D and E as well as the Lightweight CMP Profile for further information
* Broaden the scope from human users also to devices and services
* Addressed idnits feedback, specifically changing from historic LDAP V2 to LDAP V3 (RFC4510)
* Did some further editorial alignment to the XML

From version 00 -> 01:

* Performed all updates specified in CMP Updates Section 2 and Appendix A.2.
* Did some editorial allignment to the XML

Version 00:

This version consists of the text of RFC4210 with the following changes:

* Introduced the authors of this document and thanked the authors of RFC4210 for their work.
* Added a paragraph to the introduction explaining the background of this document.
* Added the change history to this appendix.

Authors' Addresses

Hendrik Brockhaus
Siemens
Werner-von-Siemens-Strasse 1

80333 Munich
Germany

Email: hendrik.brockhaus@siemens.com
URI: https://www.siemens.com

David von Oheimb
Siemens
Werner-von-Siemens-Strasse 1
80333 Munich
Germany

Email: david.von.oheimb@siemens.com
URI: https://www.siemens.com

Mike Ounsworth
Entrust
1187 Park Place
Minneapolis, MN 55379
United States of America

Email: mike.ounsworth@entrust.com
URI: https://www.entrust.com

John Gray
Entrust
1187 Park Place
Minneapolis, MN 55379
United States of America

Email: john.gray@entrust.com
URI: https://www.entrust.com