Internet Draft
PKIX Working Group
draft-ietf-pkix-cmmf-01.txt
Expires in 6 months

C. Adams (Entrust Technologies)
M. Myers (VeriSign)

July 1998

Internet X.509 Public Key Infrastructure Certificate Management Message Formats <draft-ietf-pkix-cmmf-02.txt>

1. Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of 6 months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za(Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ftp.ietf.org (US East Coast), or ftp.isi.edu (US West Coast).

2. Abstract

This is a draft of the Internet X.509 Public Key Infrastructure (PKI) Certificate Management Messages Formats (CMMF). It establishes harmonized message formats to be used in conjunction with security protocol implementations that require interaction with a Public Key Infrastructure (PKI).

This draft is being discussed on the "ietf-pkix" mailing list. To subscribe, send a message to ietf-pkix-request@tandem.com with the single word "subscribe" in the body of the message.

3. Introduction

This document defines message formats to be used between a PKI client and a PKI server or service(where a PKI is defined to encompass the roles of Registration Authority and Certification Authority). Using these message formats, a PKI client may:

- Request public key certification
    - Responses may include CA-generated key pairs
- Request revocation of a certificate
- Receive announcements directly from a PKI on:
    - CA Key Update
    - Renewed Certificates

- Subscriber Revocation notices
- CRL refresh

- Request data of a PKI, including:
    - OneÆs own or othersÆ certificate
    - CRL download

The specification of a transaction protocol using these message formats is beyond the scope of this document. It is expected that such protocols will be defined with a view towards localized interoperability needs. Where requirements exist to exchange information which is substantially aligned with the message content in this document, such protocols should reference these message formats.

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document (in uppercase, as shown) are to be interpreted as described in [RFC2119].

4. Common Data Structures

Several structures are common to more than one message format. These are:

- Status information
- Failure information
- Certificate identification
- Encrypted values

4.1 Status Information

Responses may include status information pertaining to a prior request. The following values are defined:

```
  PKIStatus ::= INTEGER {
      granted                (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
      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
      -- FullCertTemplate
```

```
  }
```

4.2 Failure Information

The following values may be used to provide more information about failure cases.

```
  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 (used for notary services)
      missingTimeStamp (8)
      -- when the timestamp is missing but should be there (by policy)
  }

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

  PKIFreeText ::= SEQUENCE SIZE (1..MAX) OF UTF8String
      -- text encoded as UTF-8 String (note:  each UTF8String SHOULD
      -- include an RFC 1766 language tag to indicate the language
      -- of the contained text)
```

4.3 Certificate Identification

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

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

4.4 Encrypted Values

Certain values in responses may require encryption beyond that that which may be performed in a PKI protocol. For example, a CA may

generate key pairs for subscribers but deliver them via an RA.As a general rule RAs should be prohibited from unnecessary or inadvertent contact with a subscriberÆs private keys.

```
EncryptedData ::= CHOICE {
      encryptedValue      [0] EncryptedValue,
      cMSEnveloped        [1] EnvelopedData }
```

The syntax of encryptedValue is defined in [CRMF]. For reference, this syntax is:

```
EncryptedValue ::= SEQUENCE {
      intendedAlg   [0] AlgorithmIdentifier  OPTIONAL,
      -- the intended algorithm for which the value will be used
      symmAlg       [1] AlgorithmIdentifier  OPTIONAL,
      -- the symmetric algorithm used to encrypt the value
      encSymmKey    [2] BIT STRING           OPTIONAL,
      -- the (encrypted) symmetric key used to encrypt the value
      keyAlg        [3] AlgorithmIdentifier  OPTIONAL,
      -- algorithm used to encrypt the symmetric key
      valueHint     [4] OCTET STRING         OPTIONAL,
      -- a brief description or identifier of the encValue content
      -- (may be meaningful only to the sending entity, and used only
      -- if EncryptedValue might be re-examined by the sending entity
      -- in the future)
        encValue       BIT STRING }
```

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

For key agreement algorithms (such as Diffie-Hellman), the encSymmKey field of EncryptedValue is not included. For key exchange algorithms (such as RSA), the encSymmKey is a symmetric key encrypted under the subscriber's public key.

If the recipient of the PKIMessage already possesses a private key usable for decryption, then the encSymmKey field MAY contain a session key encrypted using the recipientÆs public key.

5. Message Formats

5.1 Certification Request

[CRMF] defines the preferred structure for production of certification requests. It is a flexible mechanism designed to enable mandatory and optional cryptographic algorithms, centralized key generation, a variety of proof-of-possession options and several other control features useful to a robust PKI client enrollment capability.

Current industry practice can be supported for clients through use of the following structure.

```
PKCSReq ::= SEQUENCE {
    endEntityInfo    EndEntityInfo,
    regInfo          OCTET STRING OPTIONAL,
    certReqId        INTEGER OPTIONAL }
```

The endEntityInfo field is used to carry a public key, to prove possession of the public key and to provide additional ASN.1 attributes useful to production of the resulting certificate.

The certReqId field can be used to identify individual certification requests in the event multiple PKCSReqs are sent to a CA.

The regInfo field contains supplementary information related to the context of the certification request when such information is required to fulfill a certification request. This information could include subscriber contact information, billing information or other ancillary information useful to fulfillment of the certification request.

Current industry practice has established two structures for producing a signed public key. The use of explicit tagging below allows disambiguation among these options while maintaining bits on the wire compatibility with prior implementations:

```
endEntityInfo :: CHOICE {
pKCS10         CertificationRequest,
signedKey  [0] SignedPublicKeyAndChallenge }

SignedPublicKeyAndChallenge :: SEQUENCE {
publicKeyAndChallenge PublicKeyAndChallenge,
signatureAlgorithm AlgorithmIdentifier,
signature BIT STRING }

PublicKeyAndChallenge :: SEQUENCE {
spki SubjectPublicKeyInfo,
challenge IA5String }

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

The client MAY incorporate one or more standard X.509 v3 extensions in the request as an ExtensionReq attribute:

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

where Extension is imported from PKIX Part 1 and ExtensionReq is identified by the OID value {pkcs-9 14}.

5.2 Certification Response

Two options are established for the response to a certification request: a simple form that enables basic interoperability and a form that enables generation of subscriber private keys by the CA.

5.2.1 Use of SignedData

[CMS] supports a degenerate case of the SignedData content type where there are no signers on the content. This degenerate case is used to convey certificate and CRL information. Certification Authorities MAY use this format for returning certificate information resulting from the successful fulfillment of a certification request. The certificate response MUST include the actual subject certificate corresponding to the information in the certification request. The response SHOULD include other certificates which link the issuer to higher level certification authorities and corresponding CRLs.

```
CertRep  ::= SEQUENCE {
    response    SignedData, OPTIONAL
    rspInfo     OCTET STRING }
```

Successful requests would include the resulting certificate(s) in the response field along with any additional {name, value} information useful to client-side processing of a successful request. In the event of an unsuccessful request, the rspInfo field contains information regarding the failure which may be useful to the requesting entity.

RspInfo content MAY include state information the PKI client needs such as the contents of the regInfo in the CRMF (see [CRMF]). Other additional data could include detailed error information that uniquely identifies an information field supplied in the original regInfo; e.g. the Zip Code was incorrect for the specified address or locale or billing failed because credit card information was in error.

5.2.2 CertRepContent

A CertRepContent data structure contains a CA public key, a status value and optionally failure information, a subject certificate, and an encrypted private key.

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

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

```
      -- to match this response with corresponding request (a value
      -- of -1 is to be used if certReqId is not specified in the
      -- corresponding request)
```

```
      status              PKIStatusInfo,
      certifiedKeyPair    CertifiedKeyPair,   OPTIONAL }

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

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

Given an EncryptedCert and the relevant decryption key the certificate may be obtained. The purpose of this is to allow a CA to return the value of a certificate, but with the constraint that only the intended recipient can obtain the actual certificate. The benefit of this approach is that a CA may reply with a certificate even in the absence of a proof that the requester is the end entity which can use the relevant private key (note that the proof is not obtained until confirmation of receipt 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.

5.3 GetCertInitial

The GetCertInitial message is used to poll for the certificate associated with prior request if the response to that prior request was a CertRep message with a status of PENDING.

Certificates are normally referenced using the CA DN and the certificate serial number. In this case however a certificate has not yet been issued. Thus the CA and Subject DNs are present in the message as a means to identify the requested certification. The certReqId field, if used, contains a value that matches a corresponding value in a prior certification request in the event multiple requests have been sent to a CA and those requests contained a certReqId value.

```
IssuerAndSubject ::= SEQUENCE {
   issuer      Name,
   subject     Name,
   certReqId   INTEGER OPTIONAL }
```

5.4 GetCRL

This operation is used to retrieve CRLs from a certificate server or serviceÆs repository. This message is useful in circumstances where a fully-deployed Directory is either infeasible or undesired.

The GetCRL message body consists of the following syntax:

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

The Name component is the value of the Issuer DN in the subject certificate. The cRLName component may be the value of CRLDistributionPoints in the subject certificate or equivalent value in the event the certificate does not contain such a value. The time component is used by the client to specify from among potentially several issues of CRL that one whose thisUpdate value is less than but nearest to the specified time. In the absence of a time component, the CA always returns with the most recent CRL. The reasonFlags field can be used to specify from among CRLs partitioned by revocation reason. Implementors should bear in mind that while a specific revocation request has a single CRLReason code--and consequently entries in the CRL would have a single CRLReason code value--a single CRL can aggregate information for one or more reasonFlags.

5.5 Revocation Request

Two options are established to enable programmatic submission of a revocation request.

5.5.1 RevRequest

```
RevRequests ::= SEQUENCE OF RevRequest

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

For a revocation request to become a reliable object in the event of a dispute, a strong proof of originator authenticity is required. A Registration AuthorityÆs digital signature on the request can provide this proof for certificates within the scope of the RAÆs revocation authority. However, in the instance when an end-entity has lost use of their signature private key, it is impossible to produce a reliable digital signature. [CRMF] provides for the use specification of a passphrase during enrollment which may be used as an alternative authenticator in the instance of loss of use. The acceptability of this practice is a matter of local security policy.

5.5.2 RevReqContent

When requesting revocation of a certificate (or several certificates)

the following data structure is used.

```
RevReqContent ::= SEQUENCE OF RevDetails

  RevDetails ::= SEQUENCE {
      certDetails         CertTemplate,
      -- allows requester to specify as much as they can about
      -- the cert. for which revocation is requested
      -- (e.g., for cases in which serialNumber is not available)
      revocationReason    ReasonFlags,
      -- from the Draft Amendment (DAM), so CA knows what to use in
      -- Distribution point
      badSinceDate        GeneralizedTime  OPTIONAL,
      -- indicates best knowledge of sender
      crlEntryDetails     Extensions
      -- requested crlEntryExtensions }
```

5.6 Revocation Response

The response to the RevReqContent 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,
      -- 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) }
```

5.7 Challenge-Response Proof Of Possession

[CRMF] establishes the basis for a challenge-response proof of possession technique. The challenge-response messages for proof of possession of a private decryption key are specified as follows (see [MvOV97, p.404], for details).

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

  Challenge ::= SEQUENCE {
      owf                 AlgorithmIdentifier  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, where Rand is specified as
      --   Rand ::= SEQUENCE {
      --      int      INTEGER,
      --       - the randomly-generated INTEGER A (above)
      --      sender   GeneralName
      --       - the sender's name
      --   }
  }

  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.
```

It is recognized that, in general, there is a security risk associated with the decryption of arbitrary data (that is, it is typically not recommended that an entity decrypts arbitrary received ciphertext and returns the corresponding, random-looking plaintext). This is because obtaining the plaintext for carefully-chosen "ciphertexts" may in some cases leak information or provide the challenger with signatures on documents that the receiver had no intention of signing.

The witness parameter in the above challenge precludes such attacks because it convincingly demonstrates to the receiver that the challenger knew the plaintext that corresponds to each of the ciphertext challenges prior to the exchange. Therefore, in a strict information-theoretic sense, the challenger learns no information whatsoever other than the fact that the receiver is in possession of the private key corresponding to the public key for which certification is requested (which is what this protocol is explicitly designed to show). Furthermore, the challenger is not in possession of any data (e.g., a signature on a document) after the exchange that it did not already possess prior to the exchange.

5.8 PKI Confirmation content

This data structure is used in three-way protocols 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
```

5.9 Announcements

>From time to time a PKI may need to distribute information to its clients or subscribers. These messages are generally unsolicited and

and may be received by end entities at random times. Implementers of protocols that use these message formats should take this effect into account.

5.9.1 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, -- old pub signed with new priv
    newWithOld          Certificate, -- new pub signed with old priv
    newWithNew          Certificate  -- new pub signed with new priv
}
```

To change the key of the CA, the CA 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 directory and/or other means

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

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

The "new with old" certificate must have a validity period starting at the generation time of the new key pair and ending at the time by which all end entities of this CA will securely possess the new CA public key (at the latest, the expiry date of the old public key).

The "new with new" certificate must have a validity period starting at the generation time of the new key pair and ending at the time by which the CA will next update its key pair.

5.9.2 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
```

5.9.3 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
```

5.9.4 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
    -- extra CRL details(e.g., crl number, reason, location, etc.)
}
```

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.

5.10 Key recovery Response

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
}
```

5.11 PKI General Message content

>From time to time a PKI MAY announce other information not explicitly called out in this specification. In such cases the following structure MAY be used to unambiguously identify this information when transmitted as a PKI message.

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

5.12 PKI General Response content

```
GenRepContent ::= SEQUENCE OF InfoTypeAndValue
  -- The receiver is free to ignore any contained OBJ. IDs that it does
  -- not recognize.
```

5.13 Error Message

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

6. SECURITY CONSIDERATIONS

This entire memo is about security mechanisms.

One cryptographic consideration is 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 re-iterate 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.

7. References

[COR95] ISO/IEC JTC 1/SC 21, Technical Corrigendum 2 to ISO/IEC 9594-8: 1990 & 1993 (1995:E), July 1995.

[MvOV97] A. Menezes, P. van Oorschot, S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1997.

[PKCS7] RSA Laboratories, "The Public-Key Cryptography Standards (PKCS)", RSA Data Security Inc., Redwood City, California, November 1993 Release.

[PKCS10] RSA Laboratories, "The Public-Key Cryptography Standards (PKCS)", RSA Data Security Inc., Redwood City, California, November 1993 Release.

[PKCS11] RSA Laboratories, "The Public-Key Cryptography Standards - PKCS #11: Cryptographic token interface standard", RSA Data Security Inc., Redwood City, California, April 28, 1995.

[RFC2104] H. Krawczyk, M. Bellare, R. Canetti, "HMAC: Keyed Hashing for Message Authentication", Internet Request for Comments 2104, February, 1997.

[RFC2119] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", Internet Request for Comments 2119 (Best Current Practice: BCP 14), March, 1997.

[X509-AM] ISO/IEC JTC1/SC 21, Draft Amendments DAM 4 to ISO/IEC 9594-2, DAM 2 to ISO/IEC 9594-6, DAM 1 to ISO/IEC 9594-7, and DAM 1 to ISO/IEC 9594-8 on Certificate Extensions, 1 December, 1996.

[CRMF] M. Myers, C. Adams, D. Solo, D. Kemp, Certificate Request Message Format, <draft-ietf-pkix-crmf-01.txt>, May, 1998

[CMS] R. Housley, Cryptographic Message Syntax <draft-ietf-smime-cms-02.txt> December, 1997

[PKCS8] Private Key Information Syntax Standard, v1.2

Authors' Addresses
Michael Myers
VeriSign, Inc.
1390 Shorebird Way,
Mountain View, CA 94043
mmyers@verisign.com

Carlisle Adams
Entrust Technologies
750 Heron Road, Suite E08,
Ottawa, Ontario
Canada K1V 1A7
cadams@entrust.com

Appendix A: ASN.1 Module

```
CMMF DEFINITIONS IMPLICIT TAGS ::=

BEGIN

IMPORTS

      -- Directory Authentication Framework (X.509)
            Version, AlgorithmIdentifier, Name, Time,
            SubjectPublicKeyInfo, Extensions, UniqueIdentifier,
            GeneralizedTime
                FROM AuthenticationFramework { joint-iso-itu-t ds(5)
                     module(1) authenticationFramework(7) 3 }

      -- Certificate Extensions (X.509)
           GeneralName
                FROM CertificateExtensions {joint-iso-ccitt ds(5)
                     module(1) certificateExtensions(26) 0}

     -- Cryptographic Message Syntax
           EnvelopedData, SignedData
               FROM CryptographicMessageSyntax { iso(1) member-body(2)
                    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
                    modules(0) cms(1) }

     -- Certificate Request Message Format
           EncryptedValue
                FROM CRMF { . . . };

PKCSReq ::= SEQUENCE {
    endEntityInfo    EndEntityInfo,
    regInfo          OCTET STRING OPTIONAL,
    certReqId        INTEGER OPTIONAL }
```

```
CertRep  ::= SEQUENCE {
    response    SignedData, OPTIONAL
    rspInfo     OCTET STRING }
```

```
endEntityInfo :: CHOICE {
pKCS10         CertificationRequest,
signedKey  [0] SignedPublicKeyAndChallenge }

SignedPublicKeyAndChallenge :: SEQUENCE {
publicKeyAndChallenge PublicKeyAndChallenge,
signatureAlgorithm AlgorithmIdentifier,
signature BIT STRING }

PublicKeyAndChallenge :: SEQUENCE {
spki SubjectPublicKeyInfo,
challenge IA5String }

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

ExtensionReq ::= SEQUENCE OF Extension

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

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

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

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

EncryptedData ::= CHOICE {
      encryptedValue      [0] EncryptedValue,
      cMSEnveloped        [1] EnvelopedData }
```

```
IssuerAndSubject ::= SEQUENCE {
     issuer      Name,
     subject     Name,
     certReqId   INTEGER OPTIONAL }

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

RevRequests ::= SEQUENCE OF RevRequest

RevRequest ::= SEQUENCE {
    issuerName  Name,
    serialNumber  INTEGER,
    reason        CRLReason,
    passphrase    OCTET STRING OPTIONAL,
    comment       UTF8string 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)
      revocationReason    ReasonFlags,
      -- from the Draft Amendment (DAM), so CA knows what to use in
      -- Distribution point
      badSinceDate        GeneralizedTime  OPTIONAL,
      -- indicates best knowledge of sender
      crlEntryDetails     Extensions
      -- 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) }

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

Challenge ::= SEQUENCE {
      owf                 AlgorithmIdentifier  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, where Rand is specified as
      --   Rand ::= SEQUENCE {
      --      int      INTEGER,
      --       - the randomly-generated INTEGER A (above)
      --      sender   GeneralName
      --       - the sender's name
      --   }
  }

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.

PKIConfirmContent ::= NULL

PKIStatus ::= INTEGER {
      granted                (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
      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
      -- FullCertTemplate
  }

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 (used for notary services)
      missingTimeStamp (8)
      -- when the timestamp is missing but should be there (by policy)
  }

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

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

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

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

      -- include an RFC 1766 language tag to indicate the language
      -- of the contained text)

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 }

CertId ::= SEQUENCE {
```

```
      issuer           GeneralName,
      serialNumber     INTEGER }

CertAnnContent ::= Certificate
CRLAnnContent ::= SEQUENCE OF CertificateList
```

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

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 }

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

GenMsgContent ::= SEQUENCE OF InfoTypeAndValue

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