Network Working Group
Request for Comments: 4211
Obsoletes: 2511
Category: Standards Track

J. Schaad
Soaring Hawk Consulting

September 2005

Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)

Status of This Memo

Copyright Notice

Abstract

This document describes the Certificate Request Message Format (CRMF) syntax and semantics. This syntax is used to convey a request for a certificate to a Certification Authority (CA), possibly via a Registration Authority (RA), for the purposes of X.509 certificate production. The request will typically include a public key and the associated registration information. This document does not define a certificate request protocol.

Table Of Contents

1. Introduction and Terminology

This document describes the Certificate Request Message Format (CRMF). A Certificate Request Message object is used within a protocol to convey a request for a certificate to a Certification Authority (CA), possibly via a Registration Authority (RA), for the purposes of X.509 certificate production. The request will typically include a public key and the associated registration information.

The certificate request object defined in this document is not a stand-alone protocol. The information defined in this document is designed to be used by an externally defined Certificate Request Protocol (CRP). The referencing protocol is expected to define what algorithms are used, and what registration information and control structures are defined. Many of the requirements in this document refer to the referencing Certificate Request Protocol (CRP).

Certificate requests may be submitted by an RA requesting a certificate on behalf of a Subject, by a CA requesting a cross-certificate from another CA, or directly by an End Entity (EE).

The key words "MUST", "REQUIRED", "SHOULD", "RECOMMENDED", and "MAY" in this document (in uppercase, as shown) are to be interpreted as described in RFC 2119 [RFC2119].

2. Overview

Construction of a certification request involves the following steps:

a) A CertRequest object is constructed. This object may include the public key, all or a portion of the Subject name, other requested certificate fields, and additional control information related to the registration process. Depending on the CRP, this information can be specified by the Subject and potentially modified by an RA, or specified by the RA based on knowledge of the Subject or documentation presented by the Subject.

b) If required, a proof-of-possession (of the private key corresponding to the public key for which a certificate is being requested) value is calculated.

c) Additional registration information can be combined with the proof-of-possession value and the CertRequest structure to form a CertReqMessage. Additional registration information can be added by both the Subject and an RA.

d) The CertReqMessage is securely communicated to a CA. Specific means of secure transport are to be specified by each CRP that refers to this document.

2.1. Changes since RFC 2511

1. Addition of an introduction section.

2. Addition of the concept of a CRP and language relating to CRPs.

3. In section 6.2, changed regToken to authenticator.

4. Add information describing the contents of the EncryptedValue structure.

5. Changed name and contents of OID {id-regInfo 1}.

6. Added text detailing what goes into the fields of the different structures defined in the document.

7. Replaced Appendix A with a reference to [RFC2875]. The only difference is that the old text specified to use subject alt name instead of subject name if subject name was empty. This is not possible for a CA certificate issued using PKIX. It would however be useful to update RFC 2875 to have this fallback position.

7. Insert Appendix C describing why POP is necessary and what some of the different POP attacks are.

8. pop field in the CertReqMsg structure has been renamed to popo to avoid confusion between POP and pop.

9. The use of the EncryptedValue structure has been deprecated in favor of the EnvelopedData structure.

10. Add details on how private keys are to be structured when encrypted.

11. Allow for POP on key agreement algorithms other than DH.

3. CertReqMessage Syntax

A certificate request message is composed of the certificate request, an optional proof-of-possession field, and an optional registration information field.

```
CertReqMessages ::= SEQUENCE SIZE (1..MAX) OF CertReqMsg

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

The fields of CertReqMsg have the following meaning:

certReq contains the template of the certificate being requested. The template is filled in by (or on behalf of) the Subject. Not all fields within the template need to be specified. Details on this field are found in section 5.

popo contains the value used to demonstrate that the entity that will be identified as the Subject of the certificate is actually in possession of the corresponding private key. This field varies in structure and content based on the public key algorithm and the mode (encryption vs. signature) in which the algorithm is used, as specified in the KeyUsage field of the certificate to be issued. Details on this field are found in section 4.

regInfo field SHOULD contain only supplementary information relating to the context of the certificate request, where such information is required to fulfill the request. This information might include subscriber contact information, billing information, or other ancillary information useful to fulfillment of the request.

Information directly related to certificate content SHOULD be included in the certReq content. However, inclusion of additional certReq content by RAs can invalidate the popo field (depending on the details of the POP method used). Therefore, data intended for certificate content MAY be provided in regInfo.

It is the responsibility of a referencing CRP to define the details of what can be specified in the regInfo field. This document describes one method of encoding the information found in this field. Details on this encoding are found in Appendix A.

4. Proof-of-Possession (POP)

In order to prevent certain attacks (see Appendix C) and to allow a CA/RA to properly check the validity of the binding between a subject and a key pair, the PKI management structures specified here make it possible for a subject 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 CRP is free to choose how to enforce POP (e.g., out-of-band procedural means versus the CRMF in-band message) in its certification exchanges. Within a given CRP, CAs and RAs are free to choose from among the POP methods provided (i.e., this is a policy issue local to an RA/CA). A CRP SHOULD define either which POP methods are required, or specify a mechanism for clients to discover the POP methods supported.

Any CRP referencing this document MUST enforce POP by some means. 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 cannot be assumed to have been verified by the CA/RA. Therefore, one cannot truly know if the binding of the public key and the identity in the certificate is actually correct.

POP is accomplished in different ways depending on the type of key for which a certificate is requested. If a key can be used for multiple purposes (e.g., a signing and decryption RSA key), then any of the methods MAY be used. Protocol designers need to be aware that there can be hardware limitations on what POP methods may be usable, e.g., if the private key is maintained in a hardware token.

This specification allows for cases where POP is validated by the CA, the RA, or both. Some policies require the CA to verify POP during certificate issuance, in which case the RA MUST forward the end entity's CertRequest and ProofOfPossession fields unaltered to the CA. (In this case, the RA could verify the POP and reject failing certificate requests rather than forwarding them to the CA.) If the CA is not required by policy to verify POP, then the RA SHOULD forward the end entity's request and proof, unaltered, to the CA as above. If this is not possible (for example because the RA verifies POP by an out-of-band method), then the RA uses the raVerified element to attest to the CA that the required proof has been validated. If the CA/RA uses an out-of-band method to verify POP (such as physical delivery of CA/RA-generated private keys), then the ProofOfPossession field is omitted.

```
ProofOfPossession ::= CHOICE {
    raVerified        [0] NULL,
    signature         [1] POPOSigningKey,
    keyEncipherment   [2] POPOPrivKey,
    keyAgreement      [3] POPOPrivKey }
```

The fields of ProofOfPossession have the following meaning:

raVerified indicates that the RA has performed the POP required on the certificate request. This field is used by an RA when 1) the CA is not required to do its own POP verification and 2) the RA needs to change the contents of the certReq field. CRPs MUST provide a method for the RA to sign the ProofOfPossession. A requestor MUST NOT set this field and an RA/CA MUST NOT accept a ProofOfPossession where the requestor sets this field.

signature is used for performing POP with signature keys. The details of this field are covered in section 4.1.

keyEncipherment is used for performing POP with key encipherment encryption based keys (i.e., RSA). The details of this field are covered in section 4.2.

keyAgreement is used for performing POP with key agreement type encryption keys (i.e., DH). The details of this field are covered in section 4.3.

4.1. Signature Key POP

POP for a signature key is accomplished by performing a signature operation on a piece of data containing the identity for which the certificate is desired.

There are three cases that need to be looked at when doing a POP for a signature key:

1. The certificate subject has not yet established an authenticated identity with a CA/RA, but has a password and identity string from the CA/RA. In this case, the POPOSigningKeyInput structure would be filled out using the publicKeyMAC choice for authInfo, and the password and identity would be used to compute the publicKeyMAC value. The public key for the certificate being requested would be placed in both the POPOSigningKeyInput and the Certificate Template structures. The signature field is computed over the DER-encoded POPOSigningKeyInput structure.

2. The CA/RA has established an authenticated identity for the certificate subject, but the requestor is not placing it into the certificate request. In this case, the POPOSigningKeyInput structure would be filled out using the sender choice for authInfo. The public key for the certificate being requested would be placed in both the POPOSigningKeyInput and the Certificate Template structures. The signature field is computed over the DER-encoded POPOSigningKeyInput structure.

3. The certificate subject places its name in the Certificate Template structure along with the public key. In this case the poposkInput field is omitted from the POPOSigningKey structure. The signature field is computed over the DER-encoded certificate template structure.

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

The fields of POPOSigningKey have the following meaning:

poposkInput contains the data to be signed, when present. This field MUST be present when the certificate template does not contain both the public key value and a subject name value.

algorithmIdentifier identifiers the signature algorithm and an associated parameters used to produce the POP value.

signature contains the POP value produce. If poposkInput is present, the signature is computed over the DER-encoded value of poposkInput. If poposkInput is absent, the signature is computed over the DER-encoded value of certReq.

```
POPOSigningKeyInput ::= SEQUENCE {
    authInfo            CHOICE {
        sender              [0] GeneralName,
        -- used only if an authenticated identity has been
        -- established for the sender (e.g., a DN from a
        -- previously-issued and currently-valid certificate)
        publicKeyMAC        PKMACValue },
        -- used if no authenticated GeneralName currently exists for
        -- the sender; publicKeyMAC contains a password-based MAC
        -- on the DER-encoded value of publicKey
    publicKey           SubjectPublicKeyInfo }  -- from CertTemplate
```

The fields of POPOSigningKeyInput have the following meaning:

sender contains an authenticated identity that has been previously established for the subject.

publicKeyMAC contains a computed value that uses a shared secret between the CA/RA and the certificate requestor.

publicKey contains a copy of the public key from the certificate template. This MUST be exactly the same value as is contained in the certificate template.

```
PKMACValue ::= SEQUENCE {
   algId  AlgorithmIdentifier,
   value  BIT STRING }
```

The fields of PKMACValue have the following meaning:

algId identifies the algorithm used to compute the MAC value. All implementations MUST support id-PasswordBasedMAC. The details on this algorithm are presented in section 4.4.

value contains the computed MAC value. The MAC value is computed over the DER-encoded public key of the certificate subject.

The CA/RA identifies the shared secret to be used by looking at 1) the general name field in the certificate request or 2) either the regToken (see section 6.1) or authToken (see section 6.2) controls.

4.2. Key Encipherment Keys

POP for key encipherment keys is accomplished by one of three different methods. The private key can be provided to the CA/RA, an encrypted challenge from the CA/RA can be decrypted (direct method), or the created certificate can be returned encrypted and used as the challenge response (indirect method).

```
POPOPrivKey ::= CHOICE {
    thisMessage       [0] BIT STRING,   -- deprecated
    subsequentMessage [1] SubsequentMessage,
    dhMAC             [2] BIT STRING,   -- deprecated
    agreeMAC          [3] PKMACValue,
    encryptedKey      [4] EnvelopedData }
  -- for keyAgreement (only), possession is proven in this message
  -- (which contains a MAC (over the DER-encoded value of the
  -- certReq parameter in CertReqMsg, which must include both subject
  -- and publicKey) based on a key derived from the end entity's
  -- private DH key and the CA's public DH key);
  -- the dhMAC value MUST be calculated as per the directions given
  -- in RFC 2875 for static DH proof-of-possession.

SubsequentMessage ::= INTEGER {
    encrCert (0),
    challengeResp (1) }
```

The fields of POPOPrivKey have the following meaning:

thisMessage contains the encrypted private key for which a certificate is to be issued. The possession of the private key is proved by providing it to the CA/RA. This field was incorrectly

typed when the specification was first written. The correct way to use this field is to create an EncryptedValue structure where the encrypted content is the private key, the EncryptedValue structure is then wrapped in the BIT STRING type. This field has been deprecated in favor of encryptedKey.

subsequentMessage is used to indicate that the POP will be completed by decrypting a message from the CA/RA and returning a response. The type of message to be decrypted is indicated by the value used.

encrCert indicates that the certificate issued is to be returned in an encrypted form. The requestor is required to decrypt the certificate and prove success to the CA/RA. The details of this are provided by the CRP.

challengeResponse indicates that a challenge message is to be sent from the CA/RA to the requestor. The details of the challenge message and the response are to be provided by the CRP.

dhMAC is used for Diffie-Hellman key agreement keys. It contains a computed MAC that is obtained by using the requestor's private key and the CA/RA public key. The use of this field is deprecated in favor of the agreeMAC field. Details are covered in section 4.3.

agreeMAC is used for key agreement keys. It contains a computed MAC that is obtained by using the requestor's private key and a matching CA/RA public key. Details are covered in section 4.3.

macAlg contains the algorithm identifying the method used to compute the MAC value.

macValue contains the computed MAC value.

encryptedKey contains the encrypted private key matching the public key for which the certificate is to be issued. It also contains an identification value to indicate it was constructed by the requestor of the certificate. The enveloped content type MUST be id-ct-encKeyWithID.

It is expected that protocols that incorporate this specification will include the confirmation and challenge-response messages necessary for a complete protocol.

4.2.1. Private Key Info Content Type

This content type is used for 1) proving possession of private keys and 2) escrow of private keys (using the archive options control in section 6.4). This structure is based on the private key info structure from [PKCS8] but has one deliberate difference. There is a potential attack on escrow agents if they decrypt the private key but don't know to whom the encrypted key is supposed to belong. An attacker could intercept the encrypted private key, build a certificate request around it and then ask for a recovery operation on the private key.

This content type and its structure are:

```
id-ct-encKeyWithID OBJECT IDENTIFIER ::= {id-ct 21}

EncKeyWithID ::= SEQUENCE {
  privateKey           PrivateKeyInfo,
  identifier CHOICE {
    string               UTF8String,
    generalName          GeneralName
  } OPTIONAL
}

PrivateKeyInfo ::= SEQUENCE {
   version                   INTEGER,
   privateKeyAlgorithm       AlgorithmIdentifier,
   privateKey                OCTET STRING,
   attributes                [0] IMPLICIT Attributes OPTIONAL
}

Attributes ::= SET OF Attribute
```

The fields of EncKeyWithID are defined as:

privateKey contains the encoded private key. Definitions for three private key formats are included in this document. Specifications for asymmetric algorithms need to include both the public and private key definitions for consistency.

identifier contains a name that the CA/RA can associate with the requestor. This will generally be either the DN of a certificate or a text token passed and known to both the requestor and the CA/RA. This field MUST be present if the purpose is to prove possession of the private key. The field SHOULD be present if archiving a key and the archive agent is expected to decrypt the key.

The fields of PrivatekeyInfo are define as:

version MUST be the value 0

privateKeyAlgorithm contains the identifier for the private key object

privateKey is an octet string whose contents is the private key and whose format is defined by the value of privateKeyAlgorithm.

attributes is a set of attributes. They are extended information that is part of the private key information.

4.2.2. Private Key Structures

We are defining the structures here to be used for three algorithms.

4.2.2.1. D-H Private Keys

When creating a PrivateKeyInfo for a D-H key, the following rules apply:

1. The privateKeyAlgorithm MUST be set to id-dh-private-number. The parameter for id-dh-private-number is DomainParameters (imported from [PKIXALG]).

2. The ASN structure for privateKey MUST be

   DH-PrivateKey ::= INTEGER

3. The attributes field MUST be omitted.

4.2.2.2. DSA Private Keys

When creating a PrivateKeyInfo for a DSA key, the following rules apply:

1. The privateKeyAlgorithm MUST be set to id-dsa. The parameters for id-dsa is Dss-Parms (imported from [PKIXALG]).

2. The ASN structure for privateKey MUST be

   DSA-PrivateKey ::= INTEGER

3. The attributes field MUST be omitted.

4.2.2.3. RSA Private Keys

When creating a PrivateKeyInfo for an RSA key, the following rules apply:

1. The privateKeyAlgorithm MUST be set to rsaEncryption.

2. The ASN structure for privateKey MUST be RSAPrivateKey (defined in [PKCS1])

3. The attributes field MUST be omitted.

4.2.3. Challenge-Response Guidelines

The following provides guidelines to enrollment protocol authors about how an indirect proof-of-possession is expected to work and about some of the areas where one needs to be careful in crafting the messages to implement this POP method.

1. The original enrollment request includes a proof of identity of some type and the public portion of the encryption key. Note that the proof of identity needs to cover the public portion of the encryption key to prevent substitution attacks (where the attacker changes your public key for his public key).

2. The response message from the server includes an encrypted data value of some type. That value needs to be authenticated in some fashion as having come from the server. The specification needs to include the specifics of how this value is returned for the different key types. For RSA keys, the value can be specified as being directly encrypted by the RSA public key; this will not work for a D-H key where you need to specify an indirect mechanism to encrypt the value.

3. The second request message includes a hash of the decrypted value. This message MUST NOT be just the hash of the encrypted value, as one should never "sign" a completely random value. It is desirable to include information such as the identity string in the hashing process so that this can be made explicitly. This returned value MUST be included in a second proof of identity.

It is strongly suggested that transaction identifiers and nonce values be required when performing indirect POP, as this allows for 1) tying the different messages in the process together and 2) letting each entity inject some amount of random data into the process of doing identity proofs.

4.3. Key Agreement Keys

POP for key agreement keys is accomplished by one of four different methods. The first three are identical to those presented above for key encryption keys. The fourth method takes advantage of the fact that a shared secret is produced and that the value can be used to MAC information.

When the direct or indirect encryption methods presented above are used, the CA/RA will need to create an ephemeral key for those cases where the encryption algorithm parameters do not match between the CA/RA and the requestor.

The end entity may also MAC the certificate request (using a shared secret key derived from computation) as a fourth alternative for demonstrating POP. This option may be used only if the CA/RA already has a certificate that is known to the end entity and if the Subject is able to use the CA/RA's parameters.

For the DH key agreement algorithm, all implementations MUST support the static DH Proof-of-Possession. Details on this algorithm can be found in section 3 of [RFC2875]. NOTE: If either the subject or issuer name in the CA certificate is empty, then the alternative name should be used in its place.

4.4. Use of Password-Based MAC

This MAC algorithm was designed to take a shared secret (a password) and use it to compute a check value over a piece of information. The assumption is that, without the password, the correct check value cannot be computed. The algorithm computes the one-way function multiple times in order to slow down any dictionary attacks against the password value.

The algorithm identifier and parameter structure used for Password-Based MAC is:

```
id-PasswordBasedMAC OBJECT IDENTIFIER ::=
                                    { 1 2 840 113533 7 66 13}

PBMParameter ::= SEQUENCE {
   salt                OCTET STRING,
   owf                 AlgorithmIdentifier,
   iterationCount      INTEGER,
   mac                 AlgorithmIdentifier
   )
```

The fields of PEMParameter have the following meaning:

salt contains a randomly generated value used in computing the key of the MAC process. The salt SHOULD be at least 8 octets (64 bits) long.

owf identifies the algorithm and associated parameters used to compute the key used in the MAC process. All implementations MUST support SHA-1.

iterationCount identifies the number of times the hash is applied during the key computation process. The iterationCount MUST be a minimum of 100. Many people suggest using values as high as 1000 iterations as the minimum value. The trade off here is between protection of the password from attacks and the time spent by the server processing all of the different iterations in deriving passwords. Hashing is generally considered a cheap operation but this may not be true with all hash functions in the future.

mac identifies the algorithm and associated parameters of the MAC function to be used. All implementations MUST support HMAC-SHA1 [HMAC]. All implementations SHOULD support DES-MAC and Triple-DES-MAC [PKCS11].

The following is pseudo-code for the algorithm:

Inputs:
pw - an octet string containing the user's password
data - an octet string containing the value to be MAC-ed
Iter - iteration count

Output:
MAC - an octet string containing the resultant MAC value

1. Generate a random salt value S

2. Append the salt to the pw. K = pw || salt.

3. Hash the value of K. K = HASH(K)

4. If Iter is greater than zero. Iter = Iter - 1. Goto step 3.

5. Compute an HMAC as documented in [HMAC].

MAC = HASH(K XOR opad, HASH(K XOR ipad, data))

Where opad and ipad are defined in [HMAC].

5. CertRequest syntax

The CertRequest syntax consists of a request identifier, a template of certificate content, and an optional sequence of control information.

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

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

OptionalValidity ::= SEQUENCE {
   notBefore  [0] Time OPTIONAL,
   notAfter   [1] Time OPTIONAL } --at least one must be present

Time ::= CHOICE {
   utcTime        UTCTime,
   generalTime    GeneralizedTime }
```

The fields of CertRequest have the following meaning:

certReqId contains an integer value that is used by the certificate requestor to associate a specific certificate request with a certificate response.

certTemplate contains a template of an X.509 certificate. The requestor fills in those fields for which specific values are desired. Details on the fields are given below.

controls contains attributes that are not part of the certificate, but control the context in which the certificate is to be issued. Details on the controls defined in this document can be found in section 6. Other documents may define other controls. CRPs are responsible for specifying which controls are required.

The fields of CertTemplate have the following meaning:

version MUST be 2 if supplied. It SHOULD be omitted.

serialNumber MUST be omitted. This field is assigned by the CA during certificate creation.

signingAlg MUST be omitted. This field is assigned by the CA during certificate creation.

issuer is normally omitted. It would be filled in with the CA that the requestor desires to issue the certificate in situations where an RA is servicing more than one CA.

validity is normally omitted. It can be used to request that certificates either start at some point in the future or expire at some specific time. A case where this field would commonly be used is when a cross certificate is issued for a CA. In this case the validity of an existing certificate would be placed in this field so that the new certificate would have the same validity period as the existing certificate. If validity is not omitted, then at least one of the sub-fields MUST be specified. The sub-fields are as follows:

notBefore contains the requested start time of the certificate. The time follows the same rules as the notBefore time in [PROFILE].

notAfter contains the requested expiration time of the certificate. The time follows the same rules as the notAfter time in [PROFILE].

subject is filled in with the suggested name for the requestor. This would normally be filled in by a name that has been previously issued to the requestor by the CA.

publicKey contains the public key for which the certificate is being created. This field MUST be filled in if the requestor generates its own key. The field is omitted if the key is generated by the RA/CA.

issuerUID MUST be omitted. This field has been deprecated in [PROFILE].

subjectUID MUST be omitted. This field has been deprecated in [PROFILE].

extensions contains extensions that the requestor wants to have placed in the certificate. These extensions would generally deal with things such as setting the key usage to keyEncipherment.

With the exception of the publicKey field, the CA/RA is permitted to alter any requested field. The returned certificate needs to be checked by the requestor to see if the fields have been set in an acceptable manner. CA/RA SHOULD use the template fields if possible.

There are cases where all fields of the template can be omitted. If the key generation is being done at the CA/RA and the identity proof is placed in a different location (such as the id-regCtrl-regToken below), then there are no fields that need to be specified by the certificate requestor.

6. Controls Syntax

The generator of a CertRequest may include one or more control values pertaining to the processing of the request.

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

The following controls are defined by this document: regToken (section 6.1); authenticator (section 6.2); pkiPublicationInfo (section 6.3); pkiArchiveOptions (section 6.4); oldCertID (section 6.5); protocolEncrKey (section 6.6). Each CRP MUST define the set of controls supported by that protocol. Additional controls may be defined by additional RFCs or by the CRP protocol itself.

6.1. Registration Token Control

A regToken control contains one-time information (either based on a secret value or other shared information) intended to be used by the CA to verify the identity of the subject prior to issuing a certificate. Upon receipt of a certification request containing a value for regToken, the receiving CA verifies the information in order to confirm the identity claimed in the certification request.

The value for regToken may be generated by the CA and provided out of band to the subscriber, or may otherwise be available to both the CA and the subscriber. The security of any out-of-band exchange should be commensurate with the risk that the CA will tolerate with regard to accepting an intercepted value from someone other than the intended subscriber. The regToken value is not encrypted on return, if the data is considered to be sensitive, it needs to be shrouded by the requestor.

The regToken control is used only for initialization of an end entity into the PKI, whereas the authenticator control (see section 7.2) can be used for the initial as well as subsequent certification requests.

In some instances of use the value for regToken could be a text string or a numeric quantity such as a random number. In the latter case, the value is encoded as a text string representation of the binary quantity. The encoding of regToken SHALL be UTF8String.

```
id-regCtrl-regToken            OBJECT IDENTIFIER ::= { id-regCtrl 1 }
```

Without prior agreement between the subscriber and CA agents, this value would be a textual shared secret of some type. If a computed value based on that shared secret is to be used instead, it is suggested that the CRP define a new registration control for that specific computation.

6.2. Authenticator Control

An authenticator control contains information used on an ongoing basis to establish a non-cryptographic check of identity in communication with the CA. Examples include: mother's maiden name, last four digits of social security number, or other knowledge-based information shared with the subscriber's CA; a hash of such information; or other information produced for this purpose. The value for an authenticator control may be generated by the subscriber or by the CA.

In some instances of use, the value for authenticator could be a text string or a numeric quantity such as a random number. The value in the latter case is encoded as a text string representation of the binary quantity. The encoding of authenticator SHALL be UTF8String.

```
id-regCtrl-authenticator       OBJECT IDENTIFIER ::= { id-regCtrl 2 }
```

When deciding whether to use an authenticator or a regToken, use the following guidelines. If the value is a one-time usage value, then regToken would be used. If the value has a long-term usage, then the authenticator control would be used.

6.3. Publication Information Control

The pkiPublicationInfo control enables subscribers to influence the CA/RA's publication of the certificate. This control is considered advisory and can be ignored by CAs/RAs. It is defined by the following OID and syntax:

```
id-regCtrl-pkiPublicationInfo  OBJECT IDENTIFIER ::= { id-regCtrl 3 }

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

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

The fields of PKIPublicationInfo have the following meaning:

action indicates whether or not the requestor wishes the CA/RA to publish the certificate. The values and their means are:

dontPublish indicates that the requester wishes the CA/RA not to publish the certificate (this may indicate that the requester intends to publish the certificate him/herself). If dontPublish is used, the pubInfos field MUST be omitted.

pleasePublish indicates that the requestor wishes the CA/RA to publish the certificate.

pubInfos holds the locations where the requestor desires the CA/RA to publish the certificate. This field is omitted if the dontPublish choice is selected. If the requestor wants to specify some locations for the certificate to be published, and to allow the CA/RA to publish in other locations, it would specify multiple values of the SinglePubInfo structure, one of which would be dontCare.

The fields of SinglePubInfo have the following meaning:

pubMethod indicates the address type for the location at which the requestor desires the certificate to be placed by the CA/RA.

dontCare indicates that the CA/RA can publish the certificate in whatever locations it chooses. If dontCare is used, the pubInfos field MUST be omitted.

x500 indicates that the requestor wishes for the CA/RA to publish the certificate in a specific location. The location is indicated in the x500 field of pubLocation.

ldap indicates that the requestor wishes for the CA/RA to publish the certificate in a specific location. The location is indicated in the ldap field of pubLocation.

web indicates that the requestor wishes for the CA/RA to publish the certificate in a specific location. The location is indicated in the http field of pubLocation.

pubLocation contains the address at which the certificate is to be placed. The choice in the general name field is dictated by the pubMethod selection in this structure.

Publication locations can be supplied in any order. All locations are to be processed by the CA for purposes of publication.

6.4. Archive Options Control

The pkiArchiveOptions control enables subscribers to supply information needed to establish an archive of the private key corresponding to the public key of the certification request. It is defined by the following OID and syntax:

```
id-regCtrl-pkiArchiveOptions   OBJECT IDENTIFIER ::= { id-regCtrl 4 }

PKIArchiveOptions ::= CHOICE {
   encryptedPrivKey     [0] EncryptedKey,
   -- the actual value of the private key
   keyGenParameters     [1] KeyGenParameters,
   -- parameters which allow the private key to be re-generated
   archiveRemGenPrivKey [2] BOOLEAN }
   -- set to TRUE if sender wishes receiver to archive the private
   -- key of a key pair that the receiver generates in response to
   -- this request; set to FALSE if no archival is desired.

EncryptedKey ::= CHOICE {
   encryptedValue        EncryptedValue, -- deprecated
   envelopedData     [0] EnvelopedData }
   -- The encrypted private key MUST be placed in the envelopedData
   -- encryptedContentInfo encryptedContent OCTET STRING.

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 }
   -- The use of the EncryptedValue field has been deprecated in favor
   -- of the EnvelopedData structure.
   --
   -- When EncryptedValue is used to carry a private key (as opposed to
   -- a certificate), implementations MUST support the encValue field
   -- containing an encrypted PrivateKeyInfo as defined in [PKCS11],
   -- section 12.11.  If encValue contains some other format/encoding
   -- for the private key, the first octet of valueHint MAY be used
   -- to indicate the format/encoding (but note that the possible values
   -- of this octet are not specified at this time).  In all cases, the
   -- intendedAlg field MUST be used to indicate at least the OID of
   -- the intended algorithm of the private key, unless this information
   -- is known a priori to both sender and receiver by some other means.

   KeyGenParameters ::= OCTET STRING
```

The fields of PKIArchiveOptions have the following meaning:

encryptedPrivKey contains an encrypted version of the private key.

keyGenParameters contains the information needed by the requestor to regenerate the private key. As an example, for many RSA implementations one could send the first random number(s) tested for primality. The structure to go here is not defined by this document. CRPs that define content for this structure MUST define not only the content that is to go here, but also how that data is shrouded from unauthorized access.

archiveRemGenPrivKey indicates that the requestor desires that the key generated by the CA/RA on the requestor's behalf be archived.

The fields of EncryptedKey have the following meaning:

encryptedValue is longer used. This field has been deprecated along with the EncryptedValue structure.

envelopedData contains the encrypted value of the private key. CPRs that use this structure MUST define the entity or entities for whom the data is to be encrypted (the EE, escrow agents, CAs) and how that key or set of keys is to be determined. Details on constructing an EnvelopedData structure are found in [CMS]. The encrypted content MUST be an id-ct-encKeyWithID. The identifier can be omitted unless this structure is also being used to do proof-of-possession.

6.5. OldCert ID Control

If present, the OldCertID control specifies the certificate to be updated by the current certification request. The OID and syntax is:

```
id-regCtrl-oldCertID           OBJECT IDENTIFIER ::= { id-regCtrl 5 }

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

6.6. Protocol Encryption Key Control

If present, the protocolEncrKey control specifies a key that the CA is to use in encrypting a response to CertReqMessages. The OID for this control is id-regCtrl-protocolEncrKey. The parameter structure for this field is SubjectPublicKeyInfo. (This structure is defined in [PROFILE].)

```
id-regCtrl-protocolEncrKey     OBJECT IDENTIFIER ::= { id-regCtrl 6 }
```

This control is used when a CA has information to send to the subscriber that needs to be encrypted. Such information includes a private key generated by the CA for use by the subscriber.

7. RegInfo Controls

This section documents the controls that are to be placed in the regInfo field of the CertReqMsg structure.

7.1. utf8Pairs

This control is used to convey text-based information from the Subject to an RA to a CA issuing a certificate. The OID for this structure is id-regInfo-utf8Paris and has a type of UTF8String.

```
   id-regInfo-utf8Pairs    OBJECT IDENTIFIER ::= { id-regInfo 1 }
```

The name is terminated by the question mark character ('?'). The value is terminated by the percent character '%'. Name value pairs can be repeated. Thus the syntax is:

```
Name?Value%[Name?Value%]*
```

The %xx mechanism of [RFC1738] is used to encode '?' (%3f) and '%' (%25) if they are not being used for their reserved purpose. Names MUST NOT start with a numeric character.

This control can appear multiple times in the regInfo structure. Resolution of conflicts of information is a matter of local policy on the RA/CA.

Appendix A contains a set of common names and data formats corresponding to fields that commonly appear in certificates and directories.

7.2. certReq

This control is designed to deal with the problem where an RA needs to modify the certificate template proposed by a Subject, but the Subject used the certificate template as part of its POP calculation. In this case, the RA can place a new certificate template in the regInfo sequence.

This control has the OID id-regInfo-certReq and the structure CertRequest. There can only be one instance of this attribute in the regInfo sequence. If this control exists in the regInfo structure, then the certificate template in the request is ignored. The RA MUST copy all data from the core template to this attribute.

```
id-regInfo-certReq       OBJECT IDENTIFIER ::= { id-regInfo 2 }
```

8. Object Identifiers

The OID id-pkix has the value

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

-- arc for Internet X.509 PKI protocols and their components
id-pkip  OBJECT IDENTIFIER :: { id-pkix pkip(5) }

-- arc for Registration Controls in CRMF
id-regCtrl  OBJECT IDENTIFIER ::= { id-pkip regCtrl(1) }
```

```
-- arc for Registration Info in CRMF
id-regInfo       OBJECT IDENTIFIER ::= { id-pkip id-regInfo(2) }
```

9. Security Considerations

Enrollment protocols, by their very nature, involve large amounts of private information. This can include private keys, identity numbers, credit card numbers, and the like. The security of any CRP is based on the security mechanisms of the protocol and/or process used to communicate between CAs, RAs and EEs. All protocols must provide for masking, either via encryption or off-line processing, of all subscriber-sensitive information.

Many enrollment protocols provide for the initial establishment of identity between the CA/RA and the EE by the use of a token. Generally this token is delivered using an out-of-band delivery method (such as the governmental mail system). The security of any out-of-band exchange needs to be commensurate with the risk that the CA/RA will tolerate with regard to interception of the token by a third party.

Implementation must implement Proof-of-Possession (POP) values during certificate enrollment processes. A good POP algorithm needs to provide proof of two things: 1) that the key is tied to a specific user and 2) that the user has use of the key in question. Failure to implement POP allows people to create certificates where the public key and the name values do not correctly bind. This allows for impersonation on signature keys and interception of encrypted messages.

Implementations must use high entropy random number generators in producing private keys. Implementations must randomly generate content-encryption keys, message-authentication keys, initialization vectors (IVs), salt, and padding. 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, searching the resulting small set of possibilities, rather than brute force searching the whole key space. The generation of quality random numbers is difficult. RFC 4086 [RANDOM] offers important guidance in this area and Appendix 3 of FIPS Pub 186 [DSS] provides one quality PRNG technique.

Implementations must protect private keys. The compromise of a signer’s private key permits third parties to masquerade as the signer. The compromise of a decryption private key allows for interception of messages by a third party.

One feature of the certificate message request syntax is for the key generation to be performed remotely from the creation of the certificate request. This feature should never be used for generation of signing keys. If signing keys are generated for the user, then an element of repudiation comes into play. The user can claim that an item was signed by the entity that generated the key as well as any entity that might have seen the key value during transfer from the generator the to EE. Care must be taken to protect encryption keys by the remote key generator to protect against interception of the keys by a third party. This means that the encryption algorithms used need to be secure, and a content encryption key or a key encryption key must be used to mask the private key during transport back to the user. CRP protocols must never assume that a signature key generated by the user can be used to decrypt the package in which an encryption private key is transported.

This document describes a method by which key escrow may be done. There are several issues that need to be taken into account when doing key escrow. First, the client must be able to correctly identify the entity to which a key is to be escrowed or the CRP must provide a method by which the client can discover this information. A CRP cannot assume that the key escrow agent and the CA are the same entity and thus have the same names. Second, the algorithms used to mask the private key or other key generation information during transport to the escrow agent need to be commensurate with the value of the data being protected by the key. Third, the escrow agent needs to provide sufficient safeguards that an escrowed key is returned only to entities that should be able to obtain the private key. Generally, this should be restricted to the entity that escrowed the data. Fourth, the escrow data base needs to be stored in a secure manner. One common method for doing this is to re-encrypt the data to keys that only the escrow agent has access to. In this case, one may need to escrow the escrow agent key as well. Access to either the escrow agent or the archived key would amount to access to all private keys that have been escrowed with that agent.

10. References

10.1. Normative References

[PKCS1] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

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

[PKCS11] RSA Laboratories, The Public-Key Cryptography Standards - "PKCS #11 v2.11: Cryptographic Token Interface Standard", RSA Security Inc., June 2001.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[PROFILE] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[PKIXALG] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

[RFC2875] Prafullchandra, H. and J. Schaad, "Diffie-Hellman Proof-of-Possession Algorithms", RFC 2875, July 2000.

10.2. Informative References

[DSS] National Institute of Standards and Technology, FIPS Pub 186: Digital Signature Standard, May 1994.

[PKCS8] RSA Laboratories, "PKCS #8: Private-Key Information Syntax Standard", PKCS #8 v1.2, November 1993.

[RANDOM] Eastlake, D., 3rd, Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC2202] Cheng, P. and R. Glenn, "Test Cases for HMAC-MD5 and HMAC-SHA-1", RFC 2202, September 1997.

[RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

11. Acknowledgements

The working group would like to thank Michael Myers, Carlisle Adams, Dave Solo, and David Kemp, who authored the original version of this document.

The working group also gratefully acknowledges the contributions of Barbara Fox, Warwick Ford, Russ Housley, and John Pawling, whose review and comments significantly clarified and improved the utility of this specification. The members of the ca-talk mailing list also provided significant input with respect to interoperability testing.

The text of Appendix C (Why do POP) was taken from an e-mail message by Al Arsenault and was originally part of the PKIX Roadmap document.

Appendix A. Use of RegInfo for Name-Value Pairs

The "value" field of the id-regInfo-utf8Pairs string (with "tag" field equal to 12 and appropriate "length" field) will contain a series of UTF-8 name/value pairs.

This Appendix lists some common examples of such pairs for the purpose of promoting interoperability among independent implementations of this specification. It is recognized that this list is not exhaustive and will grow with time and implementation experience.

A.1. Defined Names

The following table defines a recommended set of named elements. The value in the column "Name Value" is the exact text string that will appear in the regInfo.

```
Name Value
----------
version            -- version of this variation of regInfo use
corp_company       -- company affiliation of subscriber
org_unit           -- organizational unit
mail_firstName     -- personal name component
mail_middleName    -- personal name component
mail_lastName      -- personal name component
mail_email         -- subscriber's email address
jobTitle           -- job title of subscriber
employeeID         -- employee identification number or string
mailStop           -- mail stop
issuerName         -- name of CA
subjectName        -- name of Subject
validity           -- validity interval
```

For example:

```
version?1%corp_company?Example, Inc.%org_unit?Engineering%
mail_firstName?John%mail_lastName?Smith%jobTitle?Team Leader%
mail_email?john@example.com%
```

A.2. IssuerName, SubjectName, and Validity Value Encoding

When they appear in id-regInfo-utf8Pairs syntax as named elements, the encoding of values for issuerName, subjectName, and validity SHALL use the following syntax. The characters [] indicate an optional field, ::= and | have their usual BNF meanings, and all other symbols (except spaces, which are insignificant) outside non-terminal names are terminals. Alphabetics are case-sensitive.

```
issuerName  ::= <names>
subjectName ::= <names>
<names>     ::= <name> | <names>:<name>

<validity>  ::= validity ? [<notbefore>]-[<notafter>]

<notbefore> ::= <time>
<notafter>  ::= <time>
```

Where <time> is UTC time in the form YYYYMMDD[HH[MM[SS]]]. HH, MM, and SS default to 00 and are omitted if at the and of value 00.

Example validity encoding:

```
validity?-19991231%
```

is a validity interval with no value for notBefore, and a value of December 31, 1999 for notAfter.

Each name comprises a single character name form identifier, followed by a name value of one or more UTF-8 characters. Within a name value, when it is necessary to disambiguate a character that has formatting significance at an outer level, the escape sequence %xx SHALL be used, where xx represents the hex value for the encoding concerned. The percent symbol is represented by %%.

```
<name> ::= X<xname>|O<oname>|E<ename>|D<dname>|U<uname>|I<iname>
```

Name forms and value formats are as follows:

X.500 directory name form (identifier "X"):

```
<xname> ::= <rdns>
<rdns>  ::= <rdn> | <rdns> , <rdn>
<rdn>   ::= <avas>
<avas>  ::= <ava> | <avas> + <ava>
<ava>   ::= <attyp> = <avalue>
<attyp> ::= OID.<oid> | <stdat>
```

Standard attribute type <stdat> is an alphabetic attribute type identifier from the following set:

```
C      (country)
L      (locality)
ST     (state or province)
O      (organization)
OU     (organizational unit)
CN     (common name)
STREET (street address)
E      (E-mail address).
```

<avalue> is a name component in the form of a UTF-8 character string of 1 to 64 characters, with the restriction that in the IA5 subset of UTF-8 only the characters of ASN.1 PrintableString may be used.

Other name form (identifier "O"):

```
<oname> ::= <oid> , <utf8string>
```

E-mail address (rfc822name) name form (identifier "E"):

```
<ename> ::= <ia5string>
```

DNS name form (identifier "D"):

```
<dname> ::= <ia5string>
```

URI name form (identifier "U"):

```
<uname> ::= <ia5string>
```

IP address (identifier "I"):

```
<iname> ::= <oid>
```

For example:

```
issuerName?XOU=Our CA,O=Example,C=US% subjectName?XCN=John Smith,
O=Example, C=US, E=john@example.com%
```

Appendix B. ASN.1 Structures and OIDs

```
PKIXCRMF-2005 {iso(1) identified-organization(3) dod(6) internet(1)
security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-crmf2005(36)}

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

IMPORTS
  -- Directory Authentication Framework (X.509)
     Version, AlgorithmIdentifier, Name, Time,
     SubjectPublicKeyInfo, Extensions, UniqueIdentifier, Attribute
        FROM PKIX1Explicit88 {iso(1) identified-organization(3) dod(6)
            internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
            id-pkix1-explicit(18)} -- found in [PROFILE]

  -- Certificate Extensions (X.509)
     GeneralName
        FROM PKIX1Implicit88 {iso(1) identified-organization(3) dod(6)
               internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
               id-pkix1-implicit(19)}  -- found in [PROFILE]

  -- Cryptographic Message Syntax
     EnvelopedData
        FROM CryptographicMessageSyntax2004 { iso(1) member-body(2)
             us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
             modules(0) cms-2004(24) };  -- found in [CMS]

-- The following definition may be uncommented for use with
-- ASN.1 compilers that do not understand UTF8String.

-- UTF8String ::= [UNIVERSAL 12] IMPLICIT OCTET STRING
       -- The contents of this type correspond to RFC 2279.

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

-- arc for Internet X.509 PKI protocols and their components

id-pkip  OBJECT IDENTIFIER ::= { id-pkix 5 }

id-smime OBJECT IDENTIFIER ::= { iso(1) member-body(2)
             us(840) rsadsi(113549) pkcs(1) pkcs9(9) 16 }

id-ct   OBJECT IDENTIFIER ::= { id-smime  1 }  -- content types
```

```
-- Core definitions for this module

CertReqMessages ::= SEQUENCE SIZE (1..MAX) OF CertReqMsg

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

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

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

OptionalValidity ::= SEQUENCE {
 notBefore  [0] Time OPTIONAL,
 notAfter   [1] Time OPTIONAL } -- at least one MUST be present

Controls  ::= SEQUENCE SIZE(1..MAX) OF AttributeTypeAndValue
AttributeTypeAndValue ::= SEQUENCE {
 type         OBJECT IDENTIFIER,
 value        ANY DEFINED BY type }

ProofOfPossession ::= CHOICE {
 raVerified        [0] NULL,
 -- used if the RA has already verified that the requester is in
 -- possession of the private key
 signature         [1] POPOSigningKey,
 keyEncipherment   [2] POPOPrivKey,
 keyAgreement      [3] POPOPrivKey }

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

```
 -- The signature (using "algorithmIdentifier") is on the
 -- DER-encoded value of poposkInput.  NOTE: If the CertReqMsg
 -- certReq CertTemplate contains the subject and publicKey values,
 -- then poposkInput MUST be omitted and the signature MUST be
 -- computed over the DER-encoded value of CertReqMsg certReq.  If
 -- the CertReqMsg certReq CertTemplate does not contain both the
 -- public key and subject values (i.e., if it contains only one
 -- of these, or neither), then poposkInput MUST be present and
 -- MUST be signed.


POPOSigningKeyInput ::= SEQUENCE {
 authInfo            CHOICE {
     sender              [0] GeneralName,
     -- used only if an authenticated identity has been
     -- established for the sender (e.g., a DN from a
     -- previously-issued and currently-valid certificate)
     publicKeyMAC        PKMACValue },
     -- used if no authenticated GeneralName currently exists for
     -- the sender; publicKeyMAC contains a password-based MAC
     -- on the DER-encoded value of publicKey
 publicKey           SubjectPublicKeyInfo }  -- from CertTemplate

PKMACValue ::= SEQUENCE {
algId  AlgorithmIdentifier,
-- algorithm value shall be PasswordBasedMac {1 2 840 113533 7 66 13}
-- parameter value is PBMParameter
value  BIT STRING }

PBMParameter ::= SEQUENCE {
   salt                OCTET STRING,
   owf                 AlgorithmIdentifier,
   -- AlgId for a One-Way Function (SHA-1 recommended)
   iterationCount      INTEGER,
   -- number of times the OWF is applied
   mac                 AlgorithmIdentifier
   -- the MAC AlgId (e.g., DES-MAC, Triple-DES-MAC [PKCS11],
}   -- or HMAC [HMAC, RFC2202])

POPOPrivKey ::= CHOICE {
 thisMessage       [0] BIT STRING,         -- Deprecated
 -- possession is proven in this message (which contains the private
 -- key itself (encrypted for the CA))
 subsequentMessage [1] SubsequentMessage,
 -- possession will be proven in a subsequent message
 dhMAC             [2] BIT STRING,         -- Deprecated
 agreeMAC          [3] PKMACValue,
 encryptedKey      [4] EnvelopedData }
```

```
 -- for keyAgreement (only), possession is proven in this message
 -- (which contains a MAC (over the DER-encoded value of the
 -- certReq parameter in CertReqMsg, which MUST include both subject
 -- and publicKey) based on a key derived from the end entity's
 -- private DH key and the CA's public DH key);

SubsequentMessage ::= INTEGER {
 encrCert (0),
 -- requests that resulting certificate be encrypted for the
 -- end entity (following which, POP will be proven in a
 -- confirmation message)
 challengeResp (1) }
 -- requests that CA engage in challenge-response exchange with
 -- end entity in order to prove private key possession

-- Object identifier assignments --

-- Registration Controls in CRMF
id-regCtrl OBJECT IDENTIFIER ::= { id-pkip 1 }


id-regCtrl-regToken OBJECT IDENTIFIER ::= { id-regCtrl 1 }
--with syntax:
RegToken ::= UTF8String

id-regCtrl-authenticator OBJECT IDENTIFIER ::= { id-regCtrl 2 }
--with syntax:
Authenticator ::= UTF8String

id-regCtrl-pkiPublicationInfo OBJECT IDENTIFIER ::= { id-regCtrl 3 }
--with syntax:

PKIPublicationInfo ::= SEQUENCE {
action     INTEGER {
             dontPublish (0),
             pleasePublish (1) },
pubInfos  SEQUENCE SIZE (1..MAX) OF SinglePubInfo OPTIONAL }
  -- pubInfos MUST NOT be present if action is "dontPublish"
  -- (if action is "pleasePublish" and pubInfos is omitted,
  -- "dontCare" is assumed)

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

```
id-regCtrl-pkiArchiveOptions     OBJECT IDENTIFIER ::= { id-regCtrl 4 }
--with syntax:
PKIArchiveOptions ::= CHOICE {
 encryptedPrivKey     [0] EncryptedKey,
 -- the actual value of the private key
 keyGenParameters     [1] KeyGenParameters,
 -- parameters that allow the private key to be re-generated
 archiveRemGenPrivKey [2] BOOLEAN }
 -- set to TRUE if sender wishes receiver to archive the private
 -- key of a key pair that the receiver generates in response to
 -- this request; set to FALSE if no archival is desired.

EncryptedKey ::= CHOICE {
 encryptedValue        EncryptedValue,   -- Deprecated
 envelopedData     [0] EnvelopedData }
 -- The encrypted private key MUST be placed in the envelopedData
 -- encryptedContentInfo encryptedContent OCTET STRING.

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 }
 -- the encrypted value itself
-- When EncryptedValue is used to carry a private key (as opposed to
-- a certificate), implementations MUST support the encValue field
-- containing an encrypted PrivateKeyInfo as defined in [PKCS11],
-- section 12.11.  If encValue contains some other format/encoding
-- for the private key, the first octet of valueHint MAY be used
-- to indicate the format/encoding (but note that the possible values
-- of this octet are not specified at this time).  In all cases, the
-- intendedAlg field MUST be used to indicate at least the OID of
-- the intended algorithm of the private key, unless this information
-- is known a priori to both sender and receiver by some other means.

KeyGenParameters ::= OCTET STRING
```

```
id-regCtrl-oldCertID          OBJECT IDENTIFIER ::= { id-regCtrl 5 }
--with syntax:
OldCertId ::= CertId

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

id-regCtrl-protocolEncrKey    OBJECT IDENTIFIER ::= { id-regCtrl 6 }
--with syntax:
ProtocolEncrKey ::= SubjectPublicKeyInfo

-- Registration Info in CRMF
id-regInfo OBJECT IDENTIFIER ::= { id-pkip 2 }

id-regInfo-utf8Pairs    OBJECT IDENTIFIER ::= { id-regInfo 1 }
--with syntax
UTF8Pairs ::= UTF8String


id-regInfo-certReq       OBJECT IDENTIFIER ::= { id-regInfo 2 }
--with syntax
CertReq ::= CertRequest

-- id-ct-encKeyWithID is a new content type used for CMS objects.
-- it contains both a private key and an identifier for key escrow
-- agents to check against recovery requestors.

id-ct-encKeyWithID OBJECT IDENTIFIER ::= {id-ct 21}

EncKeyWithID ::= SEQUENCE {
  privateKey           PrivateKeyInfo,
  identifier CHOICE {
    string             UTF8String,
    generalName        GeneralName
  } OPTIONAL
}

PrivateKeyInfo ::= SEQUENCE {
   version                   INTEGER,
   privateKeyAlgorithm       AlgorithmIdentifier,
   privateKey                OCTET STRING,
   attributes                [0] IMPLICIT Attributes OPTIONAL
}

Attributes ::= SET OF Attribute

END
```

Appendix C. Why do Proof-of-Possession (POP)

Proof-of-Possession, or POP, means that the CA is adequately convinced that the entity requesting a certificate for the public key Y, has access to the corresponding private key X.

POP is important because it provides an appropriate level of assurance of the correct operation of the PKI as a whole. At its lowest level, POP counters the "self-inflicted denial of service"; that is, an entity voluntarily gets a certificate that cannot be used to sign or encrypt/decrypt information. However, as the following two examples demonstrate, POP also counters less direct, but more severe, threats:

POP for signing keys: it is important to provide POP for keys used to sign material, in order to provide non-repudiation of transactions. For example, suppose Alice legitimately has private key X and its corresponding public key Y. Alice has a certificate from Charlie, a CA, containing Y. Alice uses X to sign a transaction T. Without POP, Mal could also get a certificate from Charlie containing the same public key, Y. Now, there are two possible threats: Mal could claim to have been the real signer of T; or Alice can falsely deny signing T, claiming that it was instead Mal. Since no one can reliably prove that Mal did or did not ever possess X, neither of these claims can be refuted, and thus the service provided by and the confidence in the PKI has been defeated. (Of course, if Mal really did possess X, Alice’s private key, then no POP mechanism in the world will help, but that is a different problem.)

Note that one level of protection can be gained by having Alice (as the true signer of the transaction) include in the signed information, her certificate or an identifier of her certificate (e.g., a hash of her certificate). This might make it more difficult for Mal to claim authorship; he would have to assert that he incorrectly included Alice’s certificate, rather than his own. However, it would not stop Alice from falsely repudiating her actions. Since the certificate itself is a public item, Mal indeed could have inserted Alice’s certificate or identifier into the signed transaction, and thus its presence does not indicate that Alice was the one who participated in the now-repudiated transaction. The only reliable way to stop this attack is to require that Mal prove he possesses X before his certificate is issued.

For signing keys used only for authentication, and not for non-repudiation, the threat is lower because users may not care about Alice's after-the-fact repudiation, and thus POP becomes less important. However, POP SHOULD still be done wherever feasible in this environment, by either off-line or on-line means.

POP for key management keys: Similarly, POP for key management keys (that is, keys used for either key agreement or key exchange) can help to prevent undermining confidence in the PKI. Suppose that Al is a new instructor in the Computer Science Department of a local university. Al has created a draft final exam for the Introduction to Networking course he is teaching. He wants to send a copy of the draft final to Dorothy, the Department Head, for her review prior to giving the exam. This exam will of course be encrypted, as several students have access to the computer system. However, a quick search of the certificate repository (e.g., search the repository for all records with subjectPublicKey=Dorothy's-value) turns up the fact that several students have certificates containing the same public key management key as Dorothy. At this point, if no POP has been done by the CA, Al has no way of knowing whether all of the students have simply created these certificates without knowing the corresponding private key (and thus it is safe to send the encrypted exam to Dorothy), or whether the students have somehow acquired Dorothy's private key (and thus it is certainly not safe to send the exam). Thus, the service to be provided by the PKI allowing users to communicate with one another, with confidence in who they are communicating with, has been totally defeated. If the CA is providing POP, then either no students will have such certificates, or Al can know with certainty that the students do indeed know Dorothy's private key, and act accordingly.

Author's Address

Jim Schaad
Soaring Hawk Consulting
PO Box 675
Gold Bar, WA 98251

EMail: jimsch@exmsft.com

Full Copyright Statement

Intellectual Property

Acknowledgement

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