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

M. Myers
VeriSign
C. Adams
Entrust Technologies
D. Solo
Citicorp
D. Kemp
DoD

March 1999

Internet X.509 Certificate Request Message Format

Status of this Memo

Copyright Notice

1. Abstract

This document describes the Certificate Request Message Format (CRMF). 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 associated registration information.

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.

2. Overview

Construction of a certification request involves the following steps:

a) A CertRequest value is constructed. This value may include the public key, all or a portion of the end-entity's (EE's) name, other requested certificate fields, and additional control information related to the registration process.

b) A proof of possession (of the private key corresponding to the public key for which a certificate is being requested) value may be calculated across the CertRequest value.

c) Additional registration information may be combined with the proof of possession value and the CertRequest structure to form a CertReqMessage.

d) The CertReqMessage is securely communicated to a CA. Specific means of secure transport are beyond the scope of this specification.

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,
    pop       ProofOfPossession  OPTIONAL,
    -- content depends upon key type
    regInfo   SEQUENCE SIZE(1..MAX) of AttributeTypeAndValue OPTIONAL }
```

The proof of possession field is used to demonstrate that the entity to be associated with the certificate is actually in possession of the corresponding private key. This field may be calculated across the contents of the certReq field and varies in structure and content by public key algorithm type and operational mode.

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

Information directly related to certificate content SHOULD be included in the certReq content. However, inclusion of additional certReq content by RAs may invalidate the pop field. Data therefore intended for certificate content MAY be provided in regInfo.

See Section 8 and Appendix B for example regInfo contents.

4. Proof of Possession (POP)

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

POP is accomplished in different ways depending on the type of key for which a certificate is requested. If a key can be used for multiple purposes (e.g., an RSA key) then any of the methods MAY be used.

This specification allows for cases where POP is validated by the CA, the RA, or both. Some policies may require the CA to verify POP during certification, in which case the RA MUST forward the end entity's CertRequest and ProofOfPossession fields unaltered to the CA, and as an option MAY also verify POP. 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 MAY attest to the CA that the required proof has been validated. If the CA uses an out-of-band method to verify POP (such as physical delivery of CA-generated private keys), then the ProofOfPossession field is not used.

4.1 Signature Keys

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

4.2 Key Encipherment Keys

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

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

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

4.3 Key Agreement Keys

For key agreement keys, the end entity can use any of the three methods given in Section 5.2 for encryption keys. For the direct and indirect methods, the end entity and the PKI management entity (i.e., CA or RA) must establish a shared secret key in order to prove that the end entity has possession of the private key (i.e., in order to decrypt the encrypted certificate or to construct the response to the issued challenge). Note that this need not impose any restrictions on the keys that can be certified by a given CA -- in particular, for Diffie-Hellman keys the end entity may freely choose its algorithm parameters -- provided that the CA can generate a short-term (or one-time) key pair with the appropriate parameters when necessary.

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

4.4 Proof of Possession Syntax

```
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 on the DER-encoded value of CertReqMsg certReq.  If
       -- the CertReqMsg certReq CertTemplate does not contain the public
       -- key and subject values, then poposkInput MUST be present and
       -- MUST be signed.  This strategy ensures that the public key is
       -- not present in both the poposkInput and CertReqMsg certReq
       -- CertTemplate fields.

   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,
      -- the algorithm value shall be PasswordBasedMac
      --     {1 2 840 113533 7 66 13}
      -- the parameter value is PBMParameter
      value  BIT STRING }

   POPOPrivKey ::= CHOICE {
       thisMessage       [0] BIT STRING,
       -- posession 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 }
       -- 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 Appendix A.

   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/RA engage in challenge-response exchange with
      -- end entity in order to prove private key possession
```

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

4.4.1 Use of Password-Based MAC

The following algorithm SHALL be used when publicKeyMAC is used in POPOSigningKeyInput to prove the authenticity of a request.

```
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 [RFC2104, RFC2202])
```

The process of using PBMParameter to compute publicKeyMAC and so authenticate the origin of a public key certification request consists of two stages. The first stage uses shared secret information to produce a MAC key. The second stage MACs the public key in question using this MAC key to produce an authenticated value.

Initialization of the first stage of algorithm assumes the existence of a shared secret distributed in a trusted fashion between CA/RA and end-entity. The salt value is appended to the shared secret and the one way function (owf) is applied iterationCount times, where the salted secret is the input to the first iteration and, for each successive iteration, the input is set to be the output of the previous iteration, yielding a key K.

In the second stage, K and the public key are inputs to HMAC as documented in [HMAC] to produce a value for publicKeyMAC as follows:

publicKeyMAC = Hash(K XOR opad, Hash(K XOR ipad, public key))

where ipad and opad are defined in [RFC2104].

The AlgorithmIdentifier for owf SHALL be SHA-1 {1 3 14 3 2 26} and for mac SHALL be HMAC-SHA1 {1 3 6 1 5 5 8 1 2}.

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 }
```

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 (it is recognized that this list may expand over time): regToken; authenticator; pkiPublicationInfo; pkiArchiveOptions; oldCertID; protocolEncrKey.

6.1 Registration Token Control

A regToken control contains one-time information (either based on a secret value or on knowledge) 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 of the CA accepting an intercepted value from someone other than the intended subscriber.

The regToken control would typically be used only for initialization of an end entity into the PKI, whereas the authenticator control (see Section 7.2) would typically be used for 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. The value in the latter case could be encoded either as a binary quantity or as a text string representation of the binary quantity. To ensure a uniform encoding of values regardless of the nature of the quantity, the encoding of regToken SHALL be UTF8.

6.2 Authenticator Control.

An authenticator control contains information used in 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 regToken could be a text string or a numeric quantity such as a random number. The value in the latter case could be encoded either as a binary quantity or as a text string representation of the binary quantity. To ensure a uniform encoding of values regardless of the nature of the quantity, the encoding of authenticator SHALL be UTF8.

6.3 Publication Information Control

The pkiPublicationInfo control enables subscribers to control the CA's publication of the certificate. It is defined by the following 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 }
```

If the dontPublish option is chosen, the requester indicates that the PKI should not publish the certificate (this may indicate that the requester intends to publish the certificate him/herself).

If the dontCare method is chosen, or if the PKIPublicationInfo control is omitted from the request, the requester indicates that the PKI MAY publish the certificate using whatever means it chooses.

If the requester wishes the certificate to appear in at least some locations but wishes to enable the CA to make the certificate available in other repositories, set two values of SinglePubInfo for pubInfos: one with x500, web or ldap value and one with dontCare.

The pubLocation field, if supplied, indicates where the requester would like the certificate to be found (note that the CHOICE within GeneralName includes a URL and an IP address, for example).

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 syntax:

```
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 which the receiver generates in response to
      -- this request; set to FALSE if no archival is desired.

EncryptedKey ::= CHOICE {
      encryptedValue        EncryptedValue,
      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 }

KeyGenParameters ::= OCTET STRING
```

An alternative to sending the key is to send the information about how to re-generate the key using the KeyGenParameters choice (e.g., for many RSA implementations one could send the first random numbers tested for primality). The actual syntax for this parameter may be defined in a subsequent version of this document or in another standard.

6.5 OldCert ID Control

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

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

6.6 Protocol Encryption Key Control

If present, the protocolEncrKey control specifies a key the CA is to use in encrypting a response to CertReqMessages.

This control can be 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.

The encoding of protocolEncrKey SHALL be SubjectPublicKeyInfo.

7. 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) }

-- Registration Controls in CRMF
id-regCtrl  OBJECT IDENTIFIER ::= { id-pkip regCtrl(1) }
id-regCtrl-regToken            OBJECT IDENTIFIER ::= { id-regCtrl 1 }
id-regCtrl-authenticator       OBJECT IDENTIFIER ::= { id-regCtrl 2 }
id-regCtrl-pkiPublicationInfo  OBJECT IDENTIFIER ::= { id-regCtrl 3 }
id-regCtrl-pkiArchiveOptions   OBJECT IDENTIFIER ::= { id-regCtrl 4 }
id-regCtrl-oldCertID           OBJECT IDENTIFIER ::= { id-regCtrl 5 }
id-regCtrl-protocolEncrKey     OBJECT IDENTIFIER ::= { id-regCtrl 6 }

-- Registration Info in CRMF
id-regInfo       OBJECT IDENTIFIER ::= { id-pkip id-regInfo(2) }
id-regInfo-asciiPairs    OBJECT IDENTIFIER ::= { id-regInfo 1 }
--with syntax OCTET STRING
id-regInfo-certReq       OBJECT IDENTIFIER ::= { id-regInfo 2 }
--with syntax CertRequest
```

8. Security Considerations

The security of CRMF delivery is reliant upon the security mechanisms of the protocol or process used to communicate with CAs. Such protocol or process needs to ensure the integrity, data origin authenticity, and privacy of the message. Encryption of a CRMF is strongly recommended if it contains subscriber-sensitive information and if the CA has an encryption certificate that is known to the end entity.

9. References

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

10. Acknowledgments

The authors gratefully acknowledge 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.

11. Authors' Addresses

Michael Myers
VeriSign, Inc.
1390 Shorebird Way
Mountain View, CA 94019

EMail: mmyers@verisign.com

Carlisle Adams
Entrust Technologies
750 Heron Road, Suite E08
Ottawa, Canada, K1V 1A7

EMail: cadams@entrust.com

Dave Solo
Citicorp
666 Fifth Ave, 3rd Floor
New York, Ny 10103

EMail: david.solo@citicorp.com

David Kemp
National Security Agency
Suite 6734
9800 Savage Road
Fort Meade, MD 20755

EMail: dpkemp@missi.ncsc.mil

Appendix A. Constructing "dhMAC"

This Appendix describes the method for computing the bit string "dhMAC" in the proof-of-possession POPOPrivKey structure for Diffie-Hellman certificate requests.

1. The entity generates a DH public/private key-pair.

   The DH parameters used to calculate the public SHOULD be those specified in the CA's DH certificate.

   From CA's DH certificate:
   CApub = g^x mod p (where g and p are the established DH parameters and x is the CA's private DH component)
   For entity E:
   DH private value = y
   Epub = DH public value = g^y mod p

2. The MACing process will then consist of the following steps.

a) The value of the certReq field is DER encoded, yielding a binary string. This will be the 'text' referred to in [HMAC], the data to which HMAC-SHA1 is applied.

b) A shared DH secret is computed, as follows,
shared secret = Kec = g^xy mod p

[This is done by the entity E as CApub^y and by the CA as Epub^x, where CApub is retrieved from the CA's DH certificate and Epub is retrieved from the actual certification request.]

c) A key K is derived from the shared secret Kec and the subject and issuer names in the CA's certificate as follows:

K = SHA1(DER-encoded-subjectName | Kec | DER-encoded-issuerName)

where "|" means concatenation. If subjectName in the CA certificate is an empty SEQUENCE then DER-encoded-subjectAltName should be used instead; similarly, if issuerName is an empty SEQUENCE then DER-encoded-issuerAltName should be used instead.

d) Compute HMAC-SHA1 over the data 'text' as per [RFC2104] as:
SHA1(K XOR opad, SHA1(K XOR ipad, text))

where,
opad (outer pad) = the byte 0x36 repeated 64 times
and
ipad (inner pad) = the byte 0x5C repeated 64 times.

Namely,

(1) Append zeros to the end of K to create a 64 byte string (e.g., if K is of length 16 bytes it will be appended with 48 zero bytes 0x00).
(2) XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with ipad.
(3) Append the data stream 'text' to the 64 byte string resulting from step (2).
(4) Apply SHA1 to the stream generated in step (3).
(5) XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with opad.
(6) Append the SHA1 result from step (4) to the 64 byte string resulting from step (5).
(7) Apply SHA1 to the stream generated in step (6) and output the result.

Sample code is also provided in [RFC2104, RFC2202].

e) The output of (d) is encoded as a BIT STRING (the value "dhMAC").

3. The proof-of-possession process requires the CA to carry out steps (a) through (d) and then simply compare the result of step (d) with what it received as the "dhMAC" value. If they match then the following can be concluded.

1) The Entity possesses the private key corresponding to the public key in the certification request (because it needed the private key to calculate the shared secret).

2) Only the intended CA can actually verify the request (because the CA requires its own private key to compute the same shared secret). This helps to protect from rogue CAs.

References

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

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

Acknowledgements

The details of this Appendix were provided by Hemma Prafullchandra.

Appendix B. Use of RegInfo for Name-Value Pairs

The "value" field of the id-regInfo-utf8Pairs OCTET STRING (with "tag" field equal to 12 and appropriate "length" field) will contain a series of UTF8 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.

B.1. Example Name/Value Pairs

When regInfo is used to convey one or more name-value pairs (via id-regInfo-utf8Pairs), the first and subsequent pairs SHALL be structured as follows:

```
[name?value][%name?value]*%
```

This string is then encoded into an OCTET STRING and placed into the regInfo SEQUENCE.

Reserved characters are encoded using the %xx mechanism of [RFC1738], unless they are used for their reserved purposes.

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?Acme, Inc.%org_unit?Engineering%
mail_firstName?John%mail_lastName?Smith%jobTitle?Team Leader%
mail_email?john@acme.com%
```

B.1.1. 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 UTF8 characters. Within a name value, when it is necessary to disambiguate a character which 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 UTF8 character string of 1 to 64 characters, with the restriction that in the IA5 subset of UTF8 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=Acme,C=US%
subjectName?XCN=John Smith, O=Acme, C=US, E=john@acme.com%
```

References

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

Appendix C. ASN.1 Structures and OIDs

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

CRMF DEFINITIONS IMPLICIT TAGS ::=
BEGIN

IMPORTS
     -- Directory Authentication Framework (X.509)
        Version, AlgorithmIdentifier, Name, Time,
        SubjectPublicKeyInfo, Extensions, UniqueIdentifier
           FROM PKIX1Explicit88 {iso(1) identified-organization(3) dod(6)
               internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
               id-pkix1-explicit-88(1)}

     -- 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-88(2)}

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

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

CertReqMsg ::= SEQUENCE {
    certReq   CertRequest,
    pop       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 on the DER-encoded value of CertReqMsg certReq.  If
    -- the CertReqMsg certReq CertTemplate does not contain the public
    -- key and subject values, then poposkInput MUST be present and
    -- MUST be signed.  This strategy ensures that the public key is
    -- not present in both the poposkInput and CertReqMsg certReq
    -- CertTemplate fields.

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 [RFC2104, RFC2202])

POPOPrivKey ::= CHOICE {
    thisMessage       [0] BIT STRING,
    -- posession 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 }
    -- 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 Appendix A.

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 --

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 }

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

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

-- UTF8String ::= [UNIVERSAL 12] IMPLICIT OCTET STRING

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 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 which the receiver generates in response to
```

```
    -- this request; set to FALSE if no archival is desired.

EncryptedKey ::= CHOICE {
    encryptedValue        EncryptedValue,
    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

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

END
```

Full Copyright Statement