PKIX Working Group
Internet Draft
July 13, 2001
Expires: Jan 2002

Jim Schaad
Soaring Hawk Consulting

CMC Extensions:
Server Side Key Generation and Key Archival

<draft-ietf-pkix-cmc-archive-00.txt>

Status of this Memo

Abstract

This document defines a set of extensions to [CMC] that address the desire for having two additional services: Server generation of keys, and server-side archival and subsequent recovery of key material by the server. These services are provided by the definition of additional control statements within the CMC architecture.

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

1.Requirements1. Overview

This document defines a set of extensions to [CMC] that allow for server side generation of key material and server side key archival and subsequent retrieval of the archived key material. There are some strong reasons for providing each of these services.

- Clients can have poor key generation, and having multiple clients means that this must be checked for all clients.
- Hardware tokens may not have key generation capabilities.

- End users routinely lose keys from either user error or software errors.

The creation of private keys relies on the use of good key generation algorithms. In most cases this depends on the use of a superior random number generator and strong testing for such things a prime numbers. The client side generation of this material can often times be suspect for either for reasons of poor coding in the multiple client applications, poor input (seeding material) from users and the requirements that user's not spend time waiting for things to occur. Moving the task of key generation to a server process allows for superior key generation as hardware can be installed in servers for the purpose of key generation. There is a trade off for generation of signature keys on server systems. It can provide for better key generation, but the ability for a server to know the signing key is an issue. (Note that in the case of DSS the parameter generation could be done on a server with the private key still being generated on the client's system. This allows for good parameter generation without sharing the private key with the server.)

These extensions to the CMC protocol are designed to provide the services without adding any additional round trips to the enrollment process. Server-side key generation is designed so that a client side signature key and server side key-management key can be processed in a single CMC interaction.

Sections 2 and 3 describe the concepts and structures used in transporting private keys between the server and client applications. Section 4 describes the structure and processes for server-side key generation. Section 5 describes the structure and process for doing key archival and retrieval.
-Server-side generation of keys
-Archival of private key material
-Recovery of archived private key material.

2.Protocol Overview

2. Shrouding Algorithms

Both the server-side key generation and the key recovery control attributes described in this document require that the client be able to tell the server in advance what encryption algorithm and what key value is to be used in shrouding the private keys being returned. In both of these cases the encrypted data returned is returned as either an EnvelopedData or EncryptedData object as defined by [CMS] and placed in the cmsSequence field of a ResponseBody.ContentInfo field of à.

Each request control for which the response includes encrypted data contains two fields to define type of encryption used:

The bulkEncryptionAlgId gives the OID of the bulk encryption algorithm to be used as the content encryption key (CEK) of the EnvelopedData or EncryptedData object returned. The parameters provided for this algorithm match the S/MIME capability parameters rather than the full content encryption algorithm. (This omits those parameters that are specific to a single instance of an encryption such as an initialization vector.)The parameters SHOULD be omitted if they consist of just an initialization vector and MUST be present otherwise. The initialization vector MUST be re-generated by the server for the return encryption.

The shoudIdentification field defines the method by which the server will do the key management of the CEK value in an EnvelopedData, or derive a common private key for EncryptedData. The shroudIdentification is defined as an AlgorithmIdentifier as it must identify (a) the sourceruce of the key material, (b) and the public or /salting information, and (c) the method of deriving an encryption key management key using the requested data, source key material and salt. This document defines two shroud algorithms; clients and servers MUST support id-alg-cmc-shroudPublicKey. Clients and servers SHOULD support id-alg-cmc-shroudSharedSecret.

2.1 Shroud to a Public Key

Clients can provide a public key to the server either as a bare key or wrapped in a certificate.

```
id-cmc-shroudPublicKey OBJECT IDENTIFER ::= {id-cmc XX }

ShroudPublicKey ::= CHOICE
```

```
        certificate    Certificate,
        publicKey      SubjectPublicKey
    }
```

Servers and clients MUST support use of Diffie-HellmanRSA keys for CEK key management. Servers and clients SHOULD MAY support use of RSA DH keys for CEK key management.

2.2 Shroud to a Shared-Secret Key

A shared secret value is identified to the server by the client. The derived key is then used as a KEK key in an EnvelopedData recipient info structure.

```
   id-cmc-shroudSharedSecret OBJECT IDENTIFER ::= {id-cmc XX}

   ShroudSharedSecret ::= SEQUENCE

       keyIdentifier  OCTET STRING,
       salt           OCTET STRING OPTIONAL,
       count          INTEGER DEFAULT 1,
       hash           AlgorithmIdentifier OPTIONAL,
   }
```


-- keyIdentifier provides the value to be encoded as the KEKRecipientInfo.kekid.keyIdentifier key identifierfield in the returned EncryptedData structureKEKRecipientInfo structure.

-- salt provides a salt value to be appended to the shared-secret value prior to hashing the value during key derivation

-- count provides the number of times that the hash operation is repeated during key derivation

-- hash defines the hash algorithm to be used in deriving the shared-secret key from the shared-secret value.

Clients and servers MUST support SHA-1 for the hash algorithm. Clients and servers SHOULD include salt as part of a request.

If more bits of key material are required for the encryption key that are created by the hash algorithm, subsequent key material is generated by appending one octet containing the binary value for the iteration number (1, 2, 3,à) to the shared secret followed by the salt value and then performing count hash operations.

If shared-secret shrouding is supported, clients and servers MUST support the 3DES key wrap. Clients and servers SHOULD MAY support RC2 key-wrap.

3. Private Key Info Attribute

The private key info attribute is imported from [PKCS8]. Private key information is tagged by the private key info attribute. This attribute has the ASN.1 structure:

```
id-cmc-privateKeyInfo ::= OBJECT IDENTIFER ::= {id-cmc XX20}

PrivateKeyInfo ::= SEQUENCE

   version                    INTEGER,
   privateKeyAlgorithm        AlgorithmIdentifier,
   privateKey                 OCTET STRING,
   attributes                 [0] IMPLICIT Attributes OPTIONAL
}

Attributes ::= SET OF Attribute

-- 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.  These are extended
information that is part of the private key information.
```


3.1 Private Key Structures

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

3.1.1 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 [PKIXCERTPKIXALG]).

2. The ASN structure for privateKey MUST be

DH-PrivateKey ::= INTEGER

3.3. The attributes field MUST be omitted.

3.1.2 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 [PKIXCERTPKIXALG]).

2. The ASN structure for privateKey MUST be

DSA-PrivateKey ::= INTEGER

3.3. The attributes field MUST be omitted.

3.1.3 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 aAttributes field MUST be omitted.

3.24 ContentInfo Objects for Private Key Material

The ContentInfo object that contain private key material MUST be one of EnvelopedData, EncryptedData or Data. Private key material placed in a Data ContentInfo MUST be encrypted through some other mechanism; it is beyond the scope of this document to specify that mechanism.

The inner content of the EnvelopedData or EncryptedData is a ResponseBody. The private keys are then encoded as private key info control attributes.

4. Server-Side Key Generation

This section provides the control attributes necessary for doing

server-side generation of keys for clients. The client places the request for the key generation in a request message and sends it to the server. The server will generate the key pair, create a certificate for the public key and return the data in a response message, or the server will return a failure.

Clients SHOULD NOT request server-side generation of signing-only key pairs. Servers SHOULD NOT grant requests for generation of key-pairs and creating a signing only certificate. This is designed to reduce responsibility on the server's part for possible use of the signing keys.

4.1 Server-Side Key Generation Request Attribute

The client initiates a request for server-side key generation by including the Server-Side Key Generation Request Attribute in the control attributes section of a PKIData object. The request attribute includes information about how to return the generated key as well as any client suggested items for the certificate. The control attribute for doing Server-side key generation has the following OID and structure:

```
id-cmcCtrl-ServerKeyGenRequest ::= OBJECT IDENTIFIER ::= {pkixid-
cmc XX}

ServerKeyGenRequest ::= SEQUENCE

  certificateRequest       CertTemplate,
  shroudIdentification     ShroudIdentification,
  bulkEncryptionAlgID      AlgorithmIdentifier,
  fArchiveKey              BOOLEAN DEFAULT FALSE,
  selectionCriteria        OCTET STRING OPTIONAL
}
```

-- certificateRequest contains the data fields that the client suggests for the certificate being requested for the server generated key pair.

-- shroudIdentification contains the identifier of the algorithm to be used in deriving the key used to encrypt the private key.

-- bulkEncryptionAlgId contains the encryption algorithm used in the ServerKeyGenResponse object to encrypt the private key of the server generated key pair.

-- fArchiveKey is set to TRUE if the client wishes the key to be archived as well as generated on the server. Servers MAY archive the server-generated key even if fArchiveKey is set to FALSE.

Servers SHOULD NOT generate the key pair when archival is requested but the server would not archive the key material.

-- selectionCriteria contains a string allowing for the selective retrieval of archived keys from the server. The selectionCriteria field should appear only if fArchiveKey is set to TRUE.

The client can request that the generated key be a specific algorithm by placing data in the publicKey field of the certificateRequest field. When the publicKey field is populated, the subjectPublicKey MUST be a zero length bit string and the algorithmIdentifier SHOULD omit the parameters field. If the client requests a specific algorithm, the server MUST generate a key of that algorithm (with the parameters if defined) or it MUST fail the request. Servers MUST support key generation for Diffie-Hellman RSA. Servers SHOULD MAY support key generation for Diffie-HellmanRSA. Servers MAY support key generation for other algorithms.

If the request contains no requested algorithm, servers SHOULD generate a Diffie-Hellmann RSA key exchange key pair.

A server is not required to use all of the values suggested by the client in the certificate template. Servers MUST be able to process all extensions defined in [PXIXCERT]. Servers are not required to be able to process other V3 X.509 extension transmitted using this protocol, nor are they required to be able to process other, private extensions. Servers are permitted to modify client-requested extensions. Servers MUST NOT alter an extension so as to invalidate the original intent of a client-requested extension. (For example change key usage from key exchange to signing.) If a certificate request is denied due to the inability to handle a requested extension, the server MUST respond with a failInfo attribute of unsupportedExt.

The identity proof algorithm presented in section 5.2 in [CMC] does not work if there are no certificate requests present. If the request contains no certificate request objects and the request is not being signed by a pre-existing signing certificate, the algorithm in section 5.2 should be modified to use the set of server key generation requests encoded in a sequence as the data hashed.

4.2 Server-side Key Generation Response

The server creates a server-side key generation response attribute for every key generation request made and successfully completed. The response message has a pointer to both the originating request attribute and to the body part in the current message that holds the encrypted private keys. The response message also can contain

a pointer to the certificate issued. The key recovery response control attribute has the following OID and syntax:


```
id-cmc-Ctrl-ServerKeyGenResponse ::= OBJECT IDENTIFIER ::= {pkixid-
cmc XX}

ServerKeyGenRsp ServerKeyGenResponse ::= SEQUENCE

  cmsBodyPartIdD           BodyPartID,
  requestBodyPartId             BodyPartID,
  issuerAndSerialNumber         IssuerAndSerialNumber OPTIONAL
}
```

-- cmsBodyPartIdD identifies a TaggedContentInfo contained within the enclosing PKIData. The ContentInfo contains the requested private key material.

-- requestBodyPartId contains the body part identifier for the server-side key generation request control attribute. This allows for clients to associate the resulting key and certificate with the original request.

-- issuerAndSerialNumber if present contains the identity of the certificate issued to satisfy the request. The certificate is placed in the certificate bag of the immediately encapsulating signedData object.

Clients MUST NOT assume the certificates are in any order. Servers SHOULD include all intermediate certificates needed to form complete chains to one or more self-signed certificates, not just the newly issued certificate(s). The server MAY additionally return CRLs in the CRL bag. Servers MAY include the self-signed certificates. Clients MUST NOT implicitly trust included self-signed certificate(s) merely due to its presence in the certificate bag. In the event clients receive a new self-signed certificate from the server, clients SHOULD provide a mechanism to enable the user to explicitly trust the certificate.

4.3 Control Flow

In the following control flow examples, ôclientö refers to the entity archiving a private key or requesting recovery generation of a private key, and ôserverö refers to the key archive generation facility.

1. The client creates a CMC message containing a server-side key

generation request. The required information is filled in on the request.

2. Optionally, aAny client-side key generation is done (for signing keys) and the certificate requests are constructed and placed in the PKIData request message.

3. If the client possesses a signing key, or one was created in step 2,If a signing capable key was created, it is used to sign the CMC message. Otherwise ?????If no signing key exists, the PKIData request body is placed in an AuthenticatedData structure and a shared secret is used to

authenticate the message.

4. The request is sent to the server.

5. The server does the key generation for the request.

6. The server issues all required certificates.

7. The server creates a CMC response message with the following attributes:
   a. All certificates requested are placed in the certificateList in the CMS SignedData object
   b. The private key generated is encoded as a PrivateKeyInfo object.
   c. The key object is placed in a PKI-Response as an id-cmc-privateKeyInfo control.
   d. The PKI-Response body ,is wrapped in a CMS EnvelopedData object using the shrouding information from the request.
   e. The EnvelopedData object is placed in the cmcSequence of a PKI-Respones Body.

8. The CMC response message sent to client.

4.44.4 Recovery of pre-generated keys

Some server-side key generation servers will need to limit the number of current certified key-pairs for clients. Under these circumstances a client server MAY return an already existing certified key-pair if the keys and certificate satisfy the server-side key generation request.

4.54.5 LRA behavior modifications

In many cases the actual processing of key archival and key generation controls is done not at the Certification Authority, but

at a Registration agent. This section discusses the differences in the protocol based on an RA doing the key generation.

An LRA that does the key generation operation would behave as follows:

1.1. The key generation would be done in response to the request as above.
2.2. A certificate request body is placed in a new CMC message along with the clients CMC message. (Optionally a new CMC message containing all requests plus all relevant controls could be constructed.)
3.3. The new CMC message is then sent to the Certification Authority.
4.4. When the response is returned, the LRA must builds a new CMC response message containing the encrypted private key info and all relevant certificates.

If the AuthenticatedData object is used to provide for protection of the data between the client and the RA, two different shared secrets may be needed to provide the originators identity (one for the RA and one for the CA).

5.5. Key Archival and Recovery

Servers MAY require key archival of encryption keys as a condition of issuing a certificate for that encryption key pair. Servers MAY reject all requests contained within a PKIData if any required key archival message is missing for any element within the PKIData.

There are four objects involved in key archival and recovery scenarios:

1.1. The Key Archival control attribute,
2.2. The Key Recovery Request control attribute,
3.3. The Key Recovery Response control attribute,
4.4. A ContentInfo containing private key material.

It is assumed that a key recovery operation will only return previously archived material. (That is, if an archival operation and a recovery operation happen simultaneously, the newly archived key material is not returned as part of the recovery response.)

The specification only allows for archiving of key material at the time that a certificate is requested for the key material.

The Key Recovery Response control can also be used to implement off-client generation of encryption keys. A control statement containing the required fields for key generation is generated on the client as part of the enrollment request message. This is then sent up to the server and the server responds with a Key Recovery Response containing the newly generated key. The details of the request control statement not covered in this document and would be done on a bilateral basis.

5.15.1 Key Archival Control Attribute

The key archival control attribute has the following ASN.1 syntax:

```
id-cmc-keyArchival ::= OBJECT IDENTIIFER ::= { pkixid-cmc XX12}

KeyArchival ::= SEQUENCE

  reqBodyPartID       BodyPartID,
  cmsBodyPartID       BodyPartID,
  selectionCriteria   OCTET STRING OPTIONAL
}
```

--The reqBodyPartID is a reference to the payload within the enclosing PKIData that contains the certification request for the public key component of the encryption key pair. If multiple certification requests for the same public key are included in the PKIData, reqBodyPartID may point to any one of them.

--The cmsBodyPartID is a reference to the payload within the enclosing PKIData that contains the private key to be archived. The private key MUST be the private component corresponding to the public key referenced by reqBodyPartID.

--The selectionCriteria is optional information to be associated with the archived key material to facilitate key recovery of subsections of the archive database.

5.25.2 Key Recovery Request Control Attribute

The key recovery request control attribute has the following ASN.1 syntax:

```
id-cmc-keyRecoveryReq ::= OBJECT IDENTIFER ::= {pkixid-cmc 13XX}

KeyRecoveryReq ::= SEQUENCE

  shroudIdentification     ShroudIdentification,
```

```
  bulkEncryptionAlgID      AlgorithmIdentifier,
  selectionCriteria        OCTET STRING OPTIONAL
}

ShroudIdentification ::= CHOICE

  subjectPublicKeyInfo     [0] SubjectPublicKeyInfo,
  encryptionToken          [1] AlgorithmIdentifier
}
```

--The shroudIdentification structure defines the encryption method and key material that MUST be used in the key recovery response to encrypt the recovered private key material.

--The bulkEncryptionAlgID identifies the bulk encryption algorithm that MUST be used by the server to encrypt the key material in the subsequent key recovery response.

--The selectionCriteria is optional information to be associated with the archived key material to facilitate key recovery of subsections of the archive database. If selectionCriteria is absent, all archived keys associated with the enrolling identity MUST be returned. [the selectionCriteria sounds rather detailed given the abstract quality of the rest of this discussion. Why not just present the public key(s) in question?]

Notice that the recovery request does not include any end-entity identification. Determination of the identity comes from other locations: The name in a certificate request, the name in an identity proof, the name in the shrouding certificate or the name in the signing certificate on the containing SignedInfo structure. Servers need to establish a policy to be used by that server for identifying the entity doing the request operation.

5.2.15.2.1 Key Recovery Response Control Attribute

The key recovery response control attribute has the following ASN.1 syntax:

```
  id-cmc-keyRecoveryRsp ::= OBJECT IDENTIFER ::= {pkixid-cmc 14XX}

KeyRecoveryRsp ::= SEQUENCE

  cmsBodyPartID       BodyPartID,
  keyAssocations      SEQUENCE OF KeyAssocation OPTIONAL,
  selectionCriteria        OCTET STRING OPTIONAL
}
```

```
KeyAssociation ::= SEQUENCE OF

  privateKeybodyID    BodyPartID,
  issuerAndSerialNumber IssuerAndSerialNumber
}

-- cmsBodyPartID identifies a TaggedContentInfo contained within
the enclosing PKIData.  The ContentInfo contains the requested
private key material.

-- selectionCriteria is the optional information that was used to
retrieve a subset of the keys archived in the archive database.
[see corresponding comment above]
```

5.35.3 Control Flow

In the following control flow examples, ôclientö refers to the entity archiving a private key or requesting recovery of a private key, and ôserverö refers to the key archive facility.

Control flow for Key Archival is assumed to proceed as follows:
1.1. Client retrieves an encryption certificate for the archiving server, so that key material to be archived may be encrypted to it.
2.2. Client generates an encryption key pair.
3.3. Client submits an enrollment request for the key pair. As part of the PKIData, the client includes:
a.a. A certificate request (PKCS10 or CRMF) for the key pair, which includes the public key component and a message identifier (either bodyPartID or certReqId),
b.b. A Key Archival control attribute, which includes two message identifier references:
i.i. The identifier of the certification request in (3a), and
ii.ii. The identifier of the ContentInfo in (3c).
c.c. A ContentInfo containing inside it the private key material corresponding to the public key contained in the request in (3a) and encrypted using the public key from the certificate obtained in (1).
4.4. Server receives the request, archives the key material, and issues certificates as appropriate. Server responds with an enrollment response containing the issued certificates.

It is assumed that whatever mechanisms are used to identify the entity requesting certification also serve to identify the

archiving party.

Control flow for Key Recovery is assumed to proceed as follows:
1.1. Client sends a Full PKI Request message containing the Key Recovery Request control attribute to the server. (The PKIData need contain only the Key Recovery Request attribute.)
2.2. Server performs policy-based operations to determine whether the recovery request is valid.
3.3. Assuming the request is indeed valid, the server sends back to the client a Full PKI Response containing:
a.a. One or more Key Recovery Response control attributes.
b.b. One or more Private Key attributes containing encrypted private key material as defined in section 0 above.
4.4. Client processes the response and extracts private key material from the ContentInfo(s).

The above control flow for key recovery assumes that the client possesses at least a previously-certified signature key, which is used to sign the Full PKI Request message. In the event of a catastrophic failure on the client resulting in loss of all client keys, generation and certification of a new signature key may occur simultaneously to a request for recovery of private encryption keys. Control flow for recovering from catastrophic failure may proceed as follows:
1. Client generates a new signature key pair and creates a certification request for the public component of the new signature key.
2. Client creates a Full PKI Request message containing:
a. The certificate request from Step 1,
b. A Key Recovery Request control attribute.
The Full PKI Request message is signed using the private signature key generated as part of Step 1. Following Section 4.2, the signerInfo field of the signed message will contain a NULL issuer name and a serial number corresponding to the bodyPartID of the certificate request within the PKIData. Notice that as it is not possible for the client to prove identity within the PKI (because all certified key pairs have been lost), another form of proof-of-identity is required (such as use of the identification and identityProof control attributes).
3. Client sends the Full PKI Request to the server.
4. Server performs policy-based operations on the request message to determine:
a. Whether the certification request on the new signature key should be granted, and
b. Whether the recovery request is valid.
5. Assuming that both requests are valid, the server sends back to the client a Full PKI Response containing:
a. A certificate for the signature key, corresponding to the certification request generated by the client.
b. One or more Key Recovery Response control attributes.
c. One or more Private Key attributes containing private key

material as defined in section 3.4.8.4.

6. Client processes the response and extracts private key material and certificates.

6. Additional Error Codes

7. Interactions between Server side Key Gen and Archive Retrieval

There are no interactions between the two. If a key gen uses an existing key to satisfy the key gen request, that key MUST be omitted from a key recovery request in the same CMC message. It MUST be included if the recovery request comes in a separate message.

8. Open Issues:

-- What if there were no keys to be recovered in response to a recovery response?

-- How about a first request of a key with no existing info. Can't sign the wrapper.

--

9. References

[CMS] R. Housley, "Cryptographic Message Syntax", draft-ietf-smime-cms-06.txt, June 1998

[CRMF] M. Myers, C. Adams, D. Solo, D. Kemp, "Internet X.509 Certificate Request Message Format", RFC 2511, March 1999

[DH] B. Kaliski, "PKCS 3: Diffie-Hellman Key Agreement v1.4"

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

[PKCS1] B. Kaliski, "PKCS #1: RSA Encryption, Version 1.5", RFC 2313, March 1998.

[PKCS8] RSA Laboratories, "PKCS#8: Private-Key Information Syntax Standard, Version 1.2", November 1, 1993.

[PKIXCERT] R. Housley, W. Ford, W. Polk, D. Solo "Internet X.509 Public

Key Infrastructure Certificate and CRL Profile",
RFC 2459, January 1999

[RFC 2119] "Key words for use in RFCs to Indicate Requirement
Levels", RFC 2119

10. Security Considerations

- Servers should never generate signing key material.
- POP and identity proofs are important.
- Protection of key material on user's machine, in transit and on server's machine.

Appendix A. ASN.1 Module

```
CMC-ARCHIVE

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

IMPORTS
     -- PXIX CRMF
        CertTemplate
        FROM PKIXCRMF {iso(1) identified-organization(3) dod(6)
internet(1)
              security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-
crmf(5)}

     -- PKIX CMC
        BodyPartID, id-cmc
        FROM EnrollmentMessageSyntax { iso(1) identified-
organization(3) dod(4)
               internet(1) security(5) mechansims(5) pkix(7) id-
mod(0)
               id-mod-cmc(6)}

     -- S/MIME CMS
        IssuerAndSerialNumber
        FROM CryptographicMessageSyntax { iso(1) member-body(2)
us(840)
               rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0)
cms(1) }

     -- PKIX Part 1 - Implicit
        AlgorithmIdentifier, Attribute, Certificate,
SubjectPublicKeyInfo
        FROM PKIX1Explicit88 {iso(1) identified-organization(3) dod(6)
```

```
                internet(1) security(5) mechanisms(5) pkix(7) id-
mod(0)
                id-pkix1-implicit(18)};

    id-cmc-shroudPublicKey OBJECT IDENTIFIER ::= {id-cmc XX0}
    ShroudPublicKey ::= CHOICE

        certificate     Certificate,
        publicKey       SubjectPublicKeyInfo
    }

    id-cmc-shroudSharedSecret OBJECT IDENTIFIER ::= {id-cmc XX1}
    ShroudSharedSecret ::= SEQUENCE

       keyIdentifier    OCTET STRING,
       salt             OCTET STRING OPTIONAL,
       count            INTEGER DEFAULT 1,
```



```
       hash             AlgorithmIdentifier OPTIONAL
    }

    id-cmc-privateKeyInfo OBJECT IDENTIFIER ::= {id-cmc XX2}
    PrivateKeyInfo ::= SEQUENCE

       version                  INTEGER,
       privateKeyAlgorithm      AlgorithmIdentifier,
       privateKey               OCTET STRING,
       attributes               [0] IMPLICIT Attributes OPTIONAL
    }

    Attributes ::= SET OF Attribute

    --
    -- Contains the private number for DH.  The parameters for
Algorithm
    --  identifer are DomainParameters (from rfc2459)
    --  privateKey contains DH-PrivateKey

    id-dh-private-number OBJECT IDENTIFIER ::= {id-cmc XXA9}
    DH-PrivateKey ::= INTEGER

    --
    -- privateKeyAlgorithm is id-dsa
    -- parameters is Dss-Params
    -- privateKey is DSA-PrivateKey

    DSA-PrivateKey ::= INTEGER
```

```
    -- RSA
    --
    -- privateKeyAlgorithm is rsaEncryption
    -- parameters is omitted
    -- privateKey is RSAPrivateKey

    id-cmc-ServerKeyGenRequest OBJECT IDENTIFIER ::= {id-cmc XX4}
    ServerKeyGenRequest ::= SEQUENCE

       certificateRequest       CertTemplate,
       shroudIdentification     ShroudIdentification,
       bulkEncryptionAlgID      AlgorithmIdentifier,
       fArchiveKey              BOOLEAN DEFAULT FALSE,
       selectionCriteria        OCTET STRING OPTIONAL
    }

    id-cmc-ServerKeyGenResponse OBJECT IDENTIFIER ::= {id-cmc XX5}
    ServerKeyGenResponse ::= SEQUENCE

       cmsBodyPartId            BodyPartID,
       requestBodyPartId        BodyPartID,
       issuerAndSerialNumber    IssuerAndSerialNumber OPTIONAL
    }

    id-cmc-KeyArchival OBJECT IDENTIFIER ::= {id-cmc XX6}
    KeyArchival ::= SEQUENCE

       reqBodyPartID            BodyPartID,
```



```
       cmsBodyPartID            BodyPartID,
       selectionCriteria        OCTET STRING OPTIONAL
    }

    id-cmc-keyRecoveryReq OBJECT IDENTIFIER ::= {id-cmc XX7}
    KeyRecoverReq ::= SEQUENCE

       shroudIdentification     ShroudIdentification,
       bulkEncryptionAlgID      AlgorithmIdentifier,
       selectionCriteria        OCTET STRING OPTIONAL
    }

    ShroudIdentification ::= CHOICE

       subjectPublicKeyInfo     [0] SubjectPublicKeyInfo,
       encryptionToken          [1] AlgorithmIdentifier
    }

    id-cmc-keyRecoveryRsp OBJECT IDENTIFIER ::= {id-cmc XX8}
```

```
    KeyRecoveryRsp ::= SEQUENCE

       cmsBodyPartID            BodyPartID,
       keyAssociations          SEQUENCE OF KeyAssociation OPTIONAL,
       selectionCriteria        OCTET STRING OPTIONAL
    }

    KeyAssociation ::= SEQUENCE

       privateKeybodyID         BodyPartID,
       issuerAndSerialNumber    IssuerAndSerialNumber
    }

END
```

Authors

Jim Schaad
Soaring Hawk Consulting
jimsch@exmsft.com