PKIX WORKING GROUP
Internet-Draft
Intended Status: Proposed Standard
Expires: July 5, 2012

J. Schaad
Soaring Hawk Consulting
S. Turner
IECA
P. Timmel
National Security Agency
January 2, 2012

CMC Extensions: Server Key Generation
draft-ietf-pkix-cmc-serverkeygeneration-00.txt

Abstract

This document defines a set of extensions to the Certificate Management over CMS (CMC) protocol that addresses the desire to support server-side generation of keys. This service is provided by the definition of additional control statements within the CMC architecture. Additional CMC errors are also defined.

Table of Contents

1 Introduction

This document defines a set of extensions to and errors for Certificate Management over Cryptographic Message Syntax (CMC) [RFC5272] that allows for server-side generation of key material. There are strong reasons for providing this service:

- Clients may have poor, unknown, or non-existent key generation capabilities. The creation of private keys relies on the use of good key generation algorithms and a robust random number generator. Server key generation can use specialized hardware that may not always be available on clients.

- Central storage of keys may be desired in some environments to permit key recovery. This document only addresses a request to archive server-generated keys; archival of locally generated keys and the control to retrieve archived keys is out-of-scope.

- Server key generation may be useful for provisioning keys to disconnected devices.

These extensions to the CMC protocol are designed to provide server key generation without adding any additional round trips to the enrollment process; however, additional round trips may be required based on the mechanism chosen to protect the returned key.

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 additional CMC error codes. Appendix A provides the ASN.1 module for the CMC controls and errors. Appendix B provides example encodings.

1.1 Scenarios

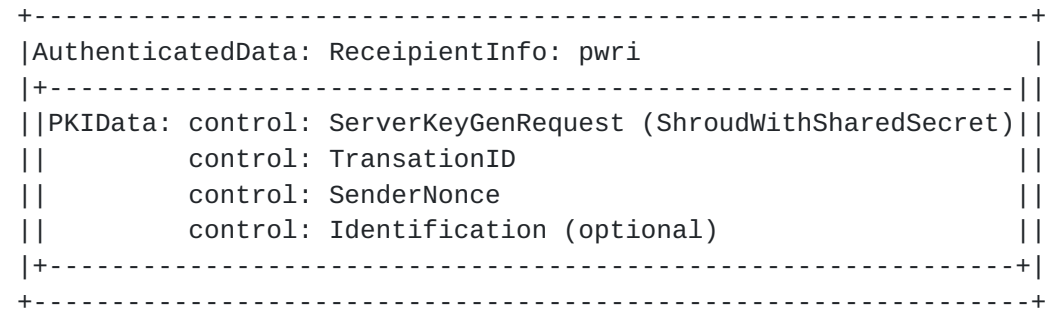
This section describes the three supported scenarios: where a shared secret is used to provide authentication and is used to encrypt the server-generated private key, where a shared secret is used for authentication and a bare, uncertified key is used for key protection, and where a certified key is used for authentication and a bare, uncertified key or certified key is used for protection. The scenarios described in the following subsections assume that the key

is generated at the Certificate Authority (CA), but key generation at Registration Authority (RA) is also supported (see Section 1.2). Also, the scenarios assume that the transaction identifier and nonce controls are used for replay protection but they are optional in [RFC5272].

1.1.1 Shared Secret for Authentication and Key Protection

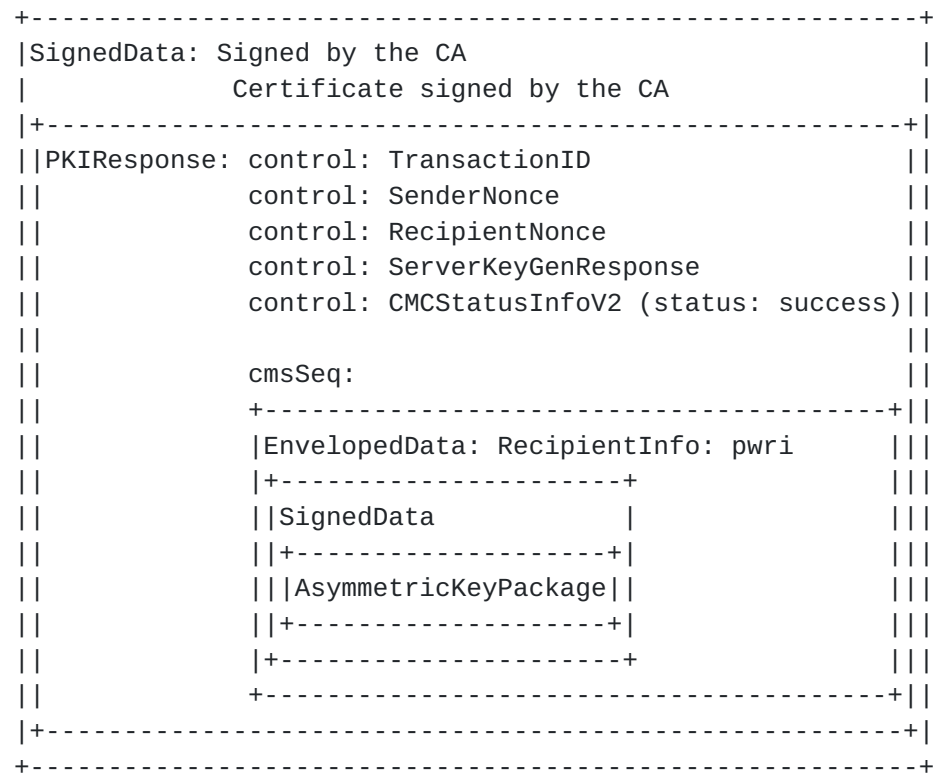
When the client uses a shared secret to authenticate to the server and requests that the server protect the response with the same shared secret, Proof-of-Possession (POP) is not necessary. POP is unnecessary in this case because the client does not need to prove to the server that it has possession of a private key because no client private key is used in the request or the response. The shared secret allows the server to authenticate the client and allows the server to encrypt the server-side generated key for the client. The shared secret is distributed via an out-of-band mechanism that is out-of-scope of this document.

When the client generates their request, they include the following control attributes in a PKIData content type [RFC5272]: ServerKeyGenRequest (see Section 4.1), TransactionID [RFC5272], SenderNonce [RFC5272], and Identification [RFC5272]. The Identification control is only needed if the ServerKeyGenRequest's certificateRequest does not include a subject. The PKIData is encapsulated in an AuthenticatedData content type and the password RecipientInfo (i.e., pwri CHOICE) is used [RFC5652]. Note that no reqSequence is included in the PKIData for the server-side key generation request. The following depicts this:



After the server authenticates the client, the server generates a response that includes the server-generated key and any associated parameters in an AsymmetricKeyPackage content type [RFC6030]. The AsymmetricKeyPackage is then encapsulated within a SignedData and that is further encapsulated within an EnvelopedData using the password RecipientInfo (i.e., pwri CHOICE). The EnvelopedData is then placed in a PKIResponse cmsSeq [RFC5272] and the following

controls are included: TransactionID, SenderNonce, RecipientNonce [RFC5272], CMCStausInfoV2 [RFC5272], and ServerKeyGenResponse (see Section 4.2). The PKIResponse is then encapsulated in a SignedData and the certificate associated with the server-generated key is placed in SignedData's certificate field [RFC5652]. The following depicts this:



The client can validate that the response came from the CA by validating the digital signature on the SignedData to a Trust Anchor (TA). After the EnvelopedData is decrypted with the shared secret, the client can also verify that the private key is associated with the public key in the returned certificate and the client can also verify that the certificate validates back to a TA.

1.1.2 Shared Secret for Authentication and Uncertified Key for Protection

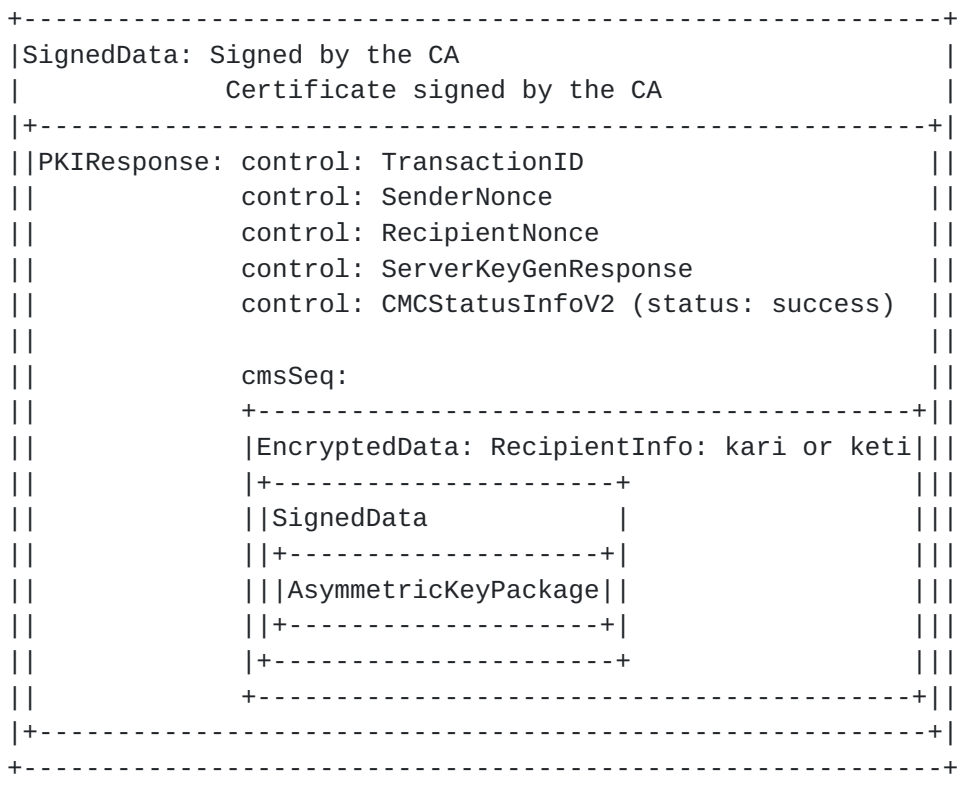
When the client uses a shared secret to authenticate to the server and requests that the server protect the response with an uncertified public key, POP is necessary because the client would like the response encrypted with a public key and the server must determine that the client actually has the private key associated with the public key. If the client requests that the response be encrypted with a key that also supports digital signatures, then the client can

provide the POP in the same transaction as the server-side key generation request. If the client requests that the response be encrypted with a key that does not support digital signatures, then additional round trips are necessary to perform POP exchanges with the encrypted/decrypted POP attributes [RFC5272].

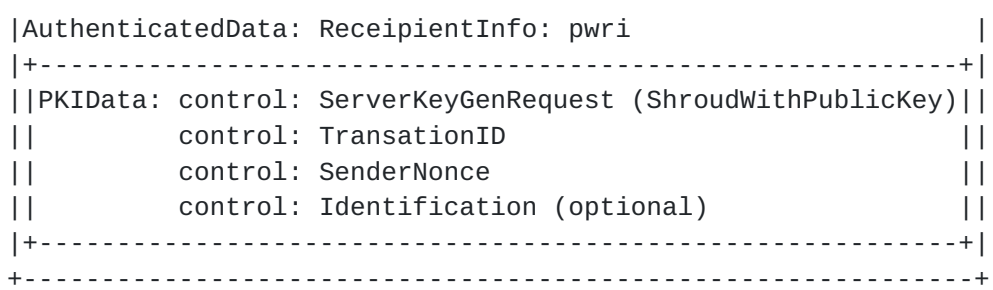
When the client's key supports both digital signatures and encryption, the client includes the following control attributes in a PKIData content type [RFC5272]: ServerKeyGenRequest control (see Section 4.1), TransactionID [RFC5272], SenderNonce [RFC5272], Identification [RFC5272], and POPLinkRandom [RFC5272]. The Identification control is only needed if the ServerKeyGenRequest's certificateRequest does not include a subject. The POPLinkRandom and the ServerKeyGenRequest'd witness values, which includes POPLinkWitnessV2, are used to prove to the server that the client has possession of the private key. The PKIData is encapsulated in an AuthenticatedData content type and the password RecipientInfo (i.e., pwri CHOICE) is used [RFC5652]. Note that no reqSequence is included for the server-side key generation request. The following depicts this:

```
+-------------------------------------------------------------+
|AuthenticatedData: ReceipientInfo: pwri                      |
|+-----------------------------------------------------------+|
||PKIData: control: ServerKeyGenRequest (ShroudWithPublicKey)||
||         control: TransationID                             ||
||         control: SenderNonce                              ||
||         control: Identification (optional)                ||
||         control: POPLinkRandom                            ||
|+-----------------------------------------------------------+|
+-------------------------------------------------------------+
```

After the server has authenticated the client and verified the POP, the server returns the server-generated key and any associated parameters in an AsymmetricKeyPackage content type [RFC6030]. The AsymmetricKeyPackage is then encapsulated within a SignedData and that is encapsulated within an EnvelopedData using the key agreement or key transports RecipientInfo (i.e., kari or ketri CHOICE). The EnvelopedData is then placed in a PKIResponse cmsSeq [RFC5272] and the following controls are included: TransactionID, SenderNonce, RecipientNonce, CMCStatusInfoV2, [RFC5272], and ServerKeyGenResponse (see Section 4.2). The PKIResponse is then encapsulated in a SignedData and the certificate associated with the server-generated key is placed in SignedData's certificate field [RFC5652]. The following depicts this:

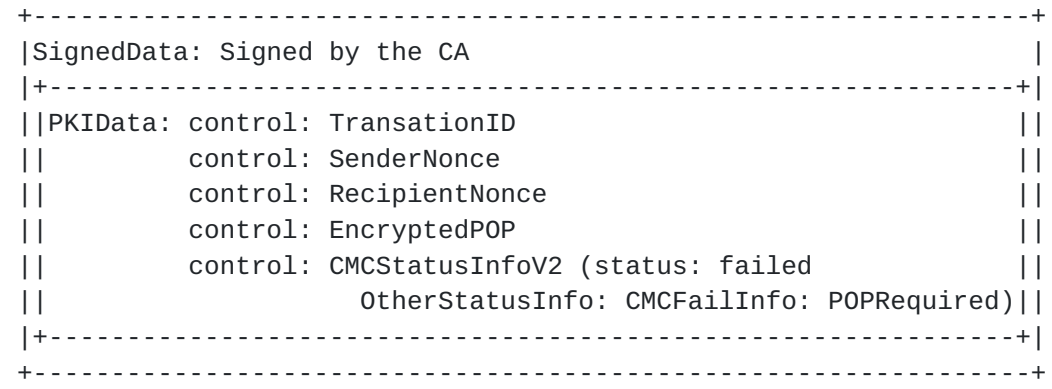


When client's key only supports encryption, the client includes the following in control attributes in a PKIData [RFC5272]: Server Key Generation Request control (see Section 4.1), TransactionID [RFC5272], SenderNonce [RFC5272], and Identification [RFC5272]. The Identification control is only needed if the certificateRequest does not include a subject. The PKIData is encapsulated in an AuthenticatedData content type and the password RecipientInfo (i.e., pwri CHOICE) is used [RFC5652]. Note that no reqSequence is included for the server-side key generation request. The following depicts this:

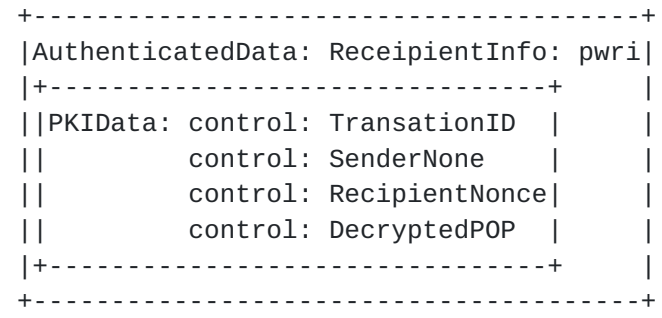


After the server has authenticated the client, the server must perform POP with the client. This requires additional round trips for the client to prove that it can decrypt date encrypted for it by the server. The server returns a PKIData with the following

controls: TransactionID, SenderNonce, EncryptedPOP, and CMCStatusInfoV2. The CMCStatusInfoV2 indicates that the request failed and why: POP required. The EncryptedPOP control attribute contains some encrypted data for the client. The PKIData is the encapsulated in a SignedData. The following depicts this:



After the client has validated the signature on the SignedData, it decrypts the EncryptedPOP with its private key and then generates a PKIData with the following controls: TransactionID, SenderNonce, RecipientNonce, and DecryptedPOP. The decrypted data is placed in the DecryptedPOP control. The PKIData is then encapsulated in an AuthenticatedData and then the client returns it to the server. The following depicts this:



After the client has authenticated the client and verified that the decrypted POP is correct, the server returns a response. It is the same response as the one returned when the client's key supports digital signatures.

1.1.3. Certificate for Authentication and Uncertified or Certified Key for Protection

When the client uses a certificate to authenticate to the server and requests that the server protect the response with a certificate or an uncertified public key or certificate, POP is still necessary

because the client would like the response encrypted with a public key and the server must determine that the client actually has the private key associated with public key. If the client requests that the response be encrypted with a key that also supports digital signatures, then the client can provide the POP in the same transaction as the server-side key generation request. If the client requests that the response be encrypted with a key that does not support digital signatures, then additional round trips are necessary to perform encrypted/decrypted POP exchanges [RFC5272]. These additional round trips are necessary for the server to determine that the client posses the private key.

The difference between this scenario and the one in Section 1.1.2 is that instead of an AuthenticatedData encapsulating the PKIData a SignedData is used.

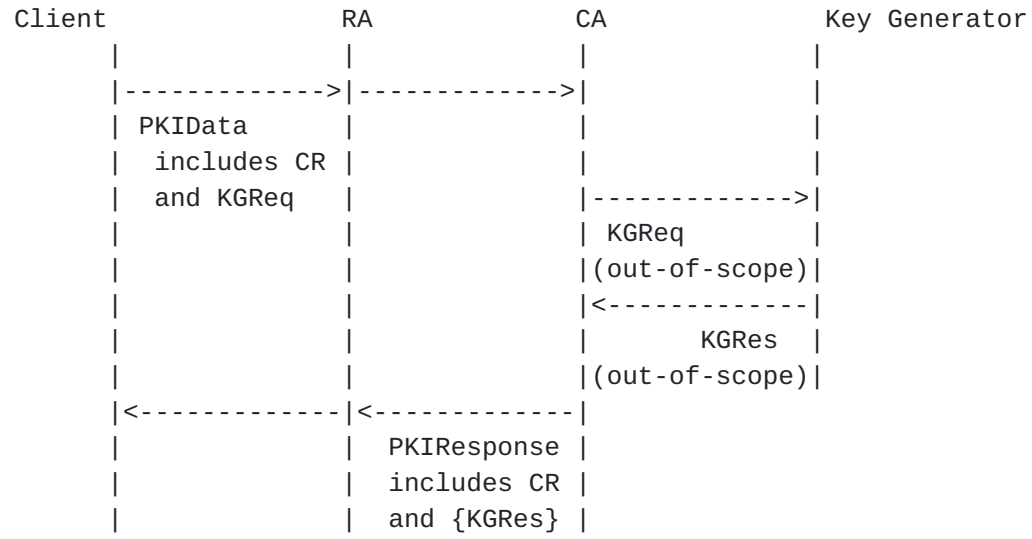
1.2. Location of Key Generator

As noted earlier, these extensions are designed to not add additional round trips to the enrollment process. This section depicts two scenarios: one when the Certification Authority (CA) generates keys and another when the Registration Authority (RA) generates the keys.

In the figures below, CR is Certificate Request/Response and KGReq is Key Generation Request, and KGRes is Key Generation Response. {} signifies encrypted.

1.2.1. CA-generated keys

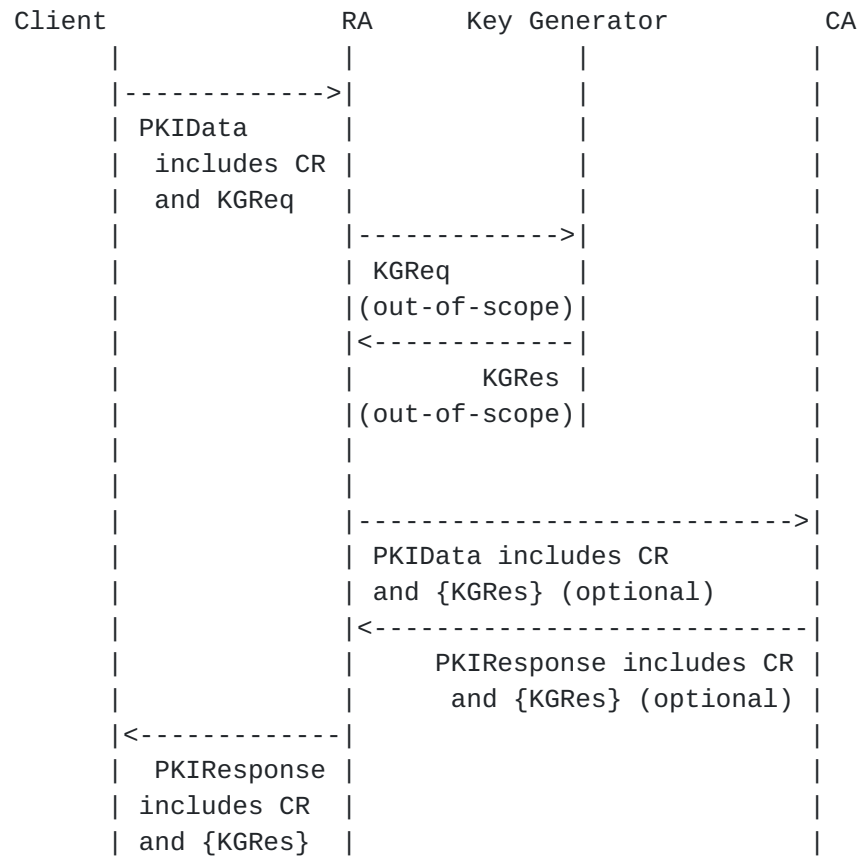
For CA-generated keys, the message flows is as follows:



In this scenario, the CA authenticates the client's request and verifies the client's identity. The CA generates the keys, issues the client's certificate, encrypts the key for the client, and signs the response. As specified in [RFC5272], the RA is optional. If an RA is involved, then the RA does not have access to encrypted portion of the response because it does not have access to either the shared secret or the client's private key.

1.2.2. RA-generated keys

For RA-generated keys, the message flow is as follows:



NOTE: The encrypted KGRes could be provided to the CA for it to return to the client or the RA can retain the encrypted KGRes and return it as part of the PKIResponse it generates. In this scenario, the RA authenticates the client's request and verifies the client's identity. The RA also verifies the POP verifies the encryption certificate (if used), generates the keys, and returns the encrypted private key. The RA also sends the certificate request to the CA and assuming it performed POP includes an indication that it did so using the LRAPOPWitness control. If the RA performed POP or based on local

policy the RA encapsulates the client's request in a SignedData and places it in a PKIData content type for the CA. The CA issues the client's certificate and signs the response. In this scenario the RA does have access to the cleartext private key but the CA does not.

1.3 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The terminology in [RFC5272] Section 2.1 applies to this profile.

1.4. Definitions

This section defines some of the terms that are used in this document:

- Identification is used as a generic term for a name, generally assigned by a server, used to match a request against a known individual. Identification can be either authenticated (a subject name in a certificate) or unauthenticated (a text string).

- Pass phrase is a shared secret string between two or more entities that allows for a MAC or encryption key to be computed. Pass phrases MUST be treated as confidential by the holders of the secret.

- Shrouding is used as a generic term to cover methods of masking the content of an object from unauthorized viewers. The most common method of shrouding used is encryption of the data.

2. Shrouding Algorithms

For the server-side key generation control attribute described in this document to function, the client must be able to tell the server in advance what encryption algorithm and what key value is to be used for encrypting the private keys being returned. In both of these cases, the encrypted data returned is returned as an EnvelopedData object as defined by [RFC5652] and placed in the cmsSequence field of a PKIResponse [RFC5272]. Client also must be able to tell the server in advance what digital signature and hash algorithms it supports to ensure the certificate request response and certificate can be verified.

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

The algCapabilities field contains the advertised capabilities of the client-side entity. This field uses the S/MIME Capabilities type defined in section 2.5.2 of [RFC5751]. The capabilities to be listed are digital signature algorithms, message digest algorithms, content encryption algorithms, key-wrap algorithms and key derivation algorithms. The shroudMethod field defines the method by which the server will do the key management of the content encrypt key (CEK) value in EnvelopedData. The shroudMethod field uses the type ShroudMethod. This type is defined as:

```
ShroudMethod ::= AlgorithmIdentifier
            { SHROUD-ALGORITHM, { ShroudAlgorithmSet } }
```

When a new shroud method is defined it MUST include (a) the source of the key material, (b) the public or salting information, and (c) the method of deriving a key management key using the requested data, source key material and salt. This document defines two shroud methods: id-cmc-shroudWithPublicKey and id-cmc-shroudWithSharedSecret. Clients and servers MUST support id-cmc-shroudWithPublicKey. Client and servers SHOULD support id-cmc-shroudWithSharedSecret.

Other shrouding methods could be defined in the future that would not involve the use of EnvelopedData. For example, one could conceive of a shrouding method that required the use of Transport Layer Security (TLS) [RFC5246] to provide the necessary security between the key generation server and the end-entity. This document does not define any such mechanism.

2.1. Shroud With a Public Key

Clients can provide a public key to the server either wrapped in a certificate or as a bare public key. For this shrouding algorithm, the public key provides the all of the information that is needed to allow for the returned key to be encrypted. The following object identifier identifies the shroudWithPublicKey shroud method:

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

shroudWithPublicKey has the ASN.1 type ShroudWithPublicKey:

```
srda-shroudWithPublicKey SHROUD-ALGORITHM ::= {
   IDENTIFIED BY id-cmc-shroudWithPublicKey,
   PARAMS TYPE ShroudWithPublicKey ARE required,
   SMIME-CAPS {IDENTIFIED BY id-cmc-shroudWithPublicKey}
}

ShroudWithPublicKey ::= CHOICE {
  certificate  Certificate,
  bareKey       SEQUENCE {
    publicKey    SubjectPublicKeyInfo,
    ski          SubjectKeyIdentifier,
    witness      SEQUENCE {
      signature    Signature,
      sigAlg       SignatureAlgorithmIdentifier,
      value        CHOICE {
          link       POPLinkWitnessV2,
          certID     ESSCertIDV2
      }
    } OPTIONAL
  }
}
```

The fields of type ShroudWithPublicKey have the following meanings:

- certificate provides a public key certificate containing the public key to be used for encrypting the server-generated private key from the server to the client. Servers SHOULD check that the subject and subject alternative names match in some way with the entity that the private key is destined for.

- bareKey allows for an arbitrary public key to be used to return the encrypted private key. The server SHOULD perform a POP on the provided key prior to using it.

  - publicKey contains the public key that is to be used for encrypting the private key returned from the server to the client.

  - ski contains the SubjectKeyIdentifier that will be used in CMS EnvelopedData to identify the public key when encrypting the private key from the server to the client.

  - witness provides a method of doing POP using a signature algorithm. This field is only present if the key is capable of doing signature operation.

- sigAlg contains the signature algorithm used to compute the signature value.
- signature contains the signature value.
- value contains the value that was signed. The value MUST be DER encoded prior to performing the signature operation.
- link is used when the outer wrapper is an AuthenticatedData structure. If this is present then the POPLinkRandom MUST be present in the list of PKIData controls. The link value is computed over the public key value using the shared secret as the MAC key.
- certID is used when the outer wrapper is a SignedData structure. It will identify the certificate that is used to sign the overall message.

When this method is used with the certificate option, the server MUST validate the certificate back to a trust anchor. Further, the server MUST check that the client provided certificate belongs to the same client that authenticated the certificate request (e.g. the certificate subjects match, or the client provided certificate belongs to the same entity as the authentication shared secret). If either of these checks fails, then the server MUST return a CMCFailInfo with the value of badCertificate, which is defined in Section 5.

When this method is used with the publicKey option and the request is authenticated using a shared secret, the server MUST verify the POPLinkRandom/POPLinkWitnessV2, as specified in [RFC5272]. If this check fails, the server MUST return a CMCFailInfo with the value of popFailed [RFC5272].

If the witness value is missing, the server MAY use the Encrypted and Decrypted POP controls [RFC5272] to perform the required POP operation. Clients MUST support this if encryption-only keys are supported.

2.2. Shroud With a Shared-Secret Key

Clients can indicate that servers use a shared secret value. For this option the key material is identified by the identifier, the key derivation algorithms supported by the client are included in the algCapabilities field. No salting material is provided by the client. The derived key is then used as a key encryption key in the EnvelopedData recipient info structure. The following object

identifier identifies the shroudWithSharedSecret control attribute:

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

shroudWithSharedSecret attribute values have the ASN.1 type ShroudWithSharedSecret:

```
shrda-shroudWithSharedSecret SHROUD-ALGORITHM ::= {
   IDENTIFIED BY id-cmc-shroudWithSharedSecret
   PARAMS TYPE ShroudWithSharedSecret ARE required
   SMIME-CAPS {IDENTIFIED BY id-cmc-shroudWithSharedSecret}

ShroudWithSharedSecret ::= UTF8String
```

The common identification string for the client and the server is placed in the ShroudWithSharedSecret field, which a UTFString [RFC5280]. In addition the client needs to place both a key derivation function and a key wrap function in the set of capabilities advertised by the client in the algCapabilities field. The identification string is used to identify the pass phrase or shared key.

When this method is used, the server MUST check that the chosen shared secret belongs to the authenticated identity of the entity that generated the certificate request. If this check fails, then the server MUST return a CMCFailInfo with the value of badSharedSecret, which is defined in Section 5. In general, while it is expected that the same identity token and shared secret used to do the identity authentication are used to derive the key encryption key this is not required.

3. Enveloping Keys

The server returns the private key and optionally the corresponding public key to the client with the AsymmetricKeyPackage content type [RFC5958]. The AsymmetricKeyPackage is encapsulated in a Cryptographic Message Syntax (CMS) EnvelopedData content type. There MUST be only one OneAsymmetricKey present in the AsymmetricKeyPackage sequence. When more than one private key is to be returned, multiple AsymmetricKeyPackages are included. The public key SHOULD be included.

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.

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 is identified by the following OID:

```
id-cmc-serverKeyGenRequest OBJECT IDENTIFIER ::= { id-cmc XX }
```

The Server-Side Key Generation Request control attribute has the following ASN.1 definition:

```
cmc-serverKeyGenRequest CMC-CONTROL ::= {
  ServerKeyGenRequest IDENTIFIED BY id-cmc-serverKeyGenRequest }

ServerKeyGenRequest ::= SEQUENCE {
  certificateRequest  CertTemplate,
  shroudMethod        ShroudMethod,
  algCapabilities     SMimeCapabilties OPTIONAL,
  archiveKey          BOOLEAN DEFAULT TRUE
}
```

The fields in ServerKeyGenRequest have the following meaning:

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

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

- algCapabilities contains the set of algorithm capabilities being advertised by the client. The server uses algorithms from this set in the ServerKeyGenResponse object to encrypt the private key of the server-generated key pair. Support is OPTIONAL because this information might be carried in a signed attribute, included within a certificate or be part of the local configuration.

- archiveKey is set to TRUE if the client wishes the key to be archived as well as generated on the server. Further processing

by the server when this is set to TRUE is out-of-scope.

The client can request that the generated key be a specific algorithm by placing data in the publicKey field of the certificateRequest field. If the publicKey field is populated, then the public key MUST be a zero length bit string. 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. If the request fails for this reason, the server SHOULD return a CMCFailInfo with a value of badAlg [RFC5272].

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 [RFC5280]. 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 digital signature.) If a certificate request is denied due to the inability to handle a requested extension, the server MUST respond with a CMCFailInfo with a value of unsupportedExt [RFC5272].

A server that does not recognize the algorithm identified in shroudMethod MUST reject the request. The server SHOULD return a CMCFailInfo with a value of badAlg [RFC5272].

A server that does not support at least one of the algCapabilities MUST reject the request. The server SHOULD return a CMCFailInfo with a value of badAlg [RFC5272].

If archiveKey is true and the server that does not support archiving of private keys MUST reject the request. The server SHOULD return a CMCFailInfo with a value of archiveNotSupported, defined in this document.

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 original 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 generation response control attribute is identified by the OID:

```
  id-cmc-serverKeyGenResponse OBJECT IDENTIFIER ::= {id-cmc XX}
```

The Server-Side Key Generation Response control attribute has the following ASN.1 definition:

```
  cmc-serverKeyGenResponse CMC-CONTROL ::= {

  ServerKeyGenResponse IDENTIFIED BY id-cmc-serverKeyGenResponse }

  ServerKeyGenResponse ::= SEQUENCE {
    cmsBodyPartId          BodyPartID,
    requestBodyPartId      BodyPartID,
    issuerAndSerialNumber  IssuerAndSerialNumber OPTIONAL
  }
```

The fields in ServerKeyGenResponse have the following meaning: o cmsBodyPartId identifies a TaggedContentInfo contained within the enclosing PKIData. The ContentInfo object is of type EnvelopedData and has an encapsulated content of id-ct-KP-aKeyPackage. As per Section 3 of this document, the body MUST contain the version, the private key algorithm identifier, and the private key and the body SHOULD contain the public key and MAY contain additional attributes.

- o 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.

- o 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) in the certificate bag. The server MAY additionally return CRLs in the CRL bag. Servers MAY include self-signed certificates. Clients MUST NOT implicitly trust included self-signed certificate(s) merely due to its presence in the certificate bag.

5. Additional Error Codes

This section defines ExtendedFailInfo errors from this document:

```
cmc-err-keyGeneration EXTENDED-FAILURE-INFO ::= {
  TYPE ErrorList IDENTIFIED BY id-tbd
}

ErrorList ::= INTEGER {
  archiveNotSupported (1),
  badCertificate (2),
  badSharedSecret (3)
}
```

If you think that you are going to want to return other information than just an error code then we can expand to a sequence of an integer and an any defined by integer or use a different extended error code just for those items.

- archiveNotSupported indicates that the server does not support archiving of private keys. The syntax for the ExtendedFailInfo is as follows:

```
cmc-err-archiveNotSupport EXTENDED-FAILURE-INFO ::={
  IDENTIFIED BY id-tbd }
```

- badCertificate indicates that the certificate to be used to encrypt the response did not validate back to a RA/CA trust anchor or the certificate does not belong to the client. The syntax for the ExtendedFailInfo is as follows:

```
cmc-err-badCertificate EXTENDED-FAILURE-INFO ::={
  IDENTIFIED BY id-tbd }
```

- badSharedSecret indicates that the shared secret provided by the client does not match that stored by the server.

```
cmc-err-badSharedSecret EXTENDED-FAILURE-INFO ::={
  IDENTIFIED BY id-tbd }
```

6. Security Considerations

Central generation of digital signature keys contains risks and is not always appropriate. Organization-specific Certificate Policies should define whether or not server-side generation of digital

signature keys is permitted.

Private keys must be appropriately protected from disclosure or modification on the server, in transit, and on the client. Cryptographic algorithms and keys used to protect the private key should be at least as strong as the private key's intended strength.

This specification requires implementations to generate key pairs and other random values. The use of inadequate pseudo-random number generators (PRNGs) can result in little or no security. The generation of quality random numbers is difficult. NIST Special Publication 800-90 [SP-800-90] and FIPS 186 [FIPS-186] offer guidance.

POP is extremely important. However, POP here is different than in client-generated key certification requests in that the POP is necessary to prove to the server that the client has possession of the private key necessary to decrypt the server-generated private key rather than possess the private key associated with the public key in the request.

Further when the shared secret is used to provide client authentication and protect the server-generated private key, the shared secret must be kept secret for the lifetime of the key. This is because disclosure could allow attackers to determine the server-generated private key. This is different than certification requests with client-generated keys because the shared secret is no longer needed after the authentication. Returning data to the wrong client is bad. Disclosure of the private key to the wrong client can result in masquerades.

7. IANA Considerations

This document makes use of object identifiers to register CMC control attributes and CMC error codes. Additionally, an object identifier is used to identify the ASN.1 module found in Appendix A. All are defined in an arc delegated by IANA to the PKIX Working Group. The current contents of the arc are located here:

http://www.imc.org/ietf-pkix/pkix-oid.asn

They were obtained by sending a request to ietf-pkix-oid-reg@imc.org. When the PKIX WG closes, this arc and registration procedures will be transferred to IANA. No further action by IANA is necessary for this document or any anticipated updates.

8. References

8.1 Normative References

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

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, June 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, August 2010.

[RFC6030] Hoyer, P., Pei, M., and S. Machani, "Portable Symmetric Key Container (PSKC)", RFC 6030, October 2010.

8.2 Informative References

[FIPS-186] National Institute of Standards and Technology (NIST), FIPS 186-3 DRAFT: Digital Signature Standard (DSS), November 2008.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[SP-800-90] National Institute of Standards and Technology (NIST), Special Publication 800-90: Recommendation for Random Number Generation Using Deterministic Random Number Bit Generators (Revised), March 2007.

Appendix A. ASN.1 Module

To be supplied later.

Appendix B. Examples

To be supplied later.

B.1. Client Requests

B.1.1. Shroud with Certificate

B.1.2. Shroud with Public Key

B.1.3. Shroud with Shared Secret

B.2. CA-Generate Key Response

B.3. RA-Generate Key Response

Authors' Addresses

Jim Schaad
Soaring Hawk Consulting

Email: jimsch@exmsft.com

Sean Turner
IECA, Inc.
3057 Nutley Street, Suite 106
Fairfax, VA 22031
USA

Email: turners@ieca.com

Paul Timmel
National Information Assurance Research Laboratory
National Security Agency

Email: pstimme@tycho.ncsc.mil