

CMS Symmetric Key Management and Distribution

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This document describes a mechanism to manage (i.e., set up, distribute, and rekey) keys used with symmetric cryptographic algorithms. Also defined herein is a mechanism to organize users into groups to support distribution of encrypted content using symmetric cryptographic algorithms. The mechanism uses the Cryptographic Message Syntax (CMS) protocol and Certificate Management over CMS (CMC) protocol to manage the symmetric keys. Any member of the group can then later use this distributed shared key to decrypt other CMS encrypted objects with the symmetric key. This mechanism has been developed to support Secure/Multipurpose Internet Mail Extensions (S/MIME) Mail List Agents (MLAs).

Table of Contents

1. Introduction	4
1.1. Conventions Used in This Document	4
1.2. Applicability to E-mail	5
1.3. Applicability to Repositories	5
1.4. Using the Group Key	5
2. Architecture	6
3. Protocol Interactions	7
3.1. Control Attributes	8
3.1.1. GL Use KEK	10
3.1.2. Delete GL	14
3.1.3. Add GL Member	14
3.1.4. Delete GL Member	15
3.1.5. Rekey GL	16
3.1.6. Add GL Owner	16
3.1.7. Remove GL Owner	17
3.1.8. GL Key Compromise	17
3.1.9. GL Key Refresh	18
3.1.10. GLA Query Request and Response	18
3.1.10.1. GLA Query Request	18
3.1.10.2. GLA Query Response	19
3.1.10.3. Request and Response Types	19
3.1.11. Provide Cert	19
3.1.12. Update Cert	20
3.1.13. GL Key	21
3.2. Use of CMC, CMS, and PKIX	23
3.2.1. Protection Layers	23
3.2.1.1. Minimum Protection	23
3.2.1.2. Additional Protection	24
3.2.2. Combining Requests and Responses	24
3.2.3. GLA Generated Messages	26
3.2.4. CMC Control Attributes and CMS Signed Attributes	27
3.2.4.1. Using cMCStatusInfoExt	27
3.2.4.2. Using transactionId	30
3.2.4.3. Using Nonces and signingTime	30
3.2.4.4. CMC and CMS Attribute Support Requirements	31
3.2.5. Resubmitted GL Member Messages	31
3.2.6. PKIX Certificate and CRL Profile	31
4. Administrative Messages	32
4.1. Assign KEK to GL	32
4.2. Delete GL from GLA	36
4.3. Add Members to GL	38
4.3.1. GLO Initiated Additions	39
4.3.2. Prospective Member Initiated Additions	47
4.4. Delete Members from GL	49
4.4.1. GLO Initiated Deletions	50

4.4.2. Member Initiated Deletions	56
4.5. Request Rekey of GL	57
4.5.1. GLO Initiated Rekey Requests	59
4.5.2. GLA Initiated Rekey Requests	62
4.6. Change GLO	63
4.7. Indicate KEK Compromise	65
4.7.1. GL Member Initiated KEK Compromise Message	66
4.7.2. GLO Initiated KEK Compromise Message	67
4.8. Request KEK Refresh	69
4.9. GLA Query Request and Response	70
4.10. Update Member Certificate	73
4.10.1. GLO and GLA Initiated Update Member Certificate ...	73
4.10.2. GL Member Initiated Update Member Certificate	75
5. Distribution Message	77
5.1. Distribution Process	78
6. Algorithms	79
6.1. KEK Generation Algorithm	79
6.2. Shared KEK Wrap Algorithm	79
6.3. Shared KEK Algorithm	79
7. Message Transport	80
8. Security Considerations	80
9. Acknowledgements	81
10. References	81
10.1. Normative References	81
10.2. Informative References	82
Appendix A. ASN.1 Module	83

1. Introduction

With the ever-expanding use of secure electronic communications (e.g., S/MIME [MSG]), users require a mechanism to distribute encrypted data to multiple recipients (i.e., a group of users). There are essentially two ways to encrypt the data for recipients: using asymmetric algorithms with public key certificates (PKCs) or symmetric algorithms with symmetric keys.

With asymmetric algorithms, the originator forms an originator-determined content-encryption key (CEK) and encrypts the content, using a symmetric algorithm. Then, using an asymmetric algorithm and the recipient's PKCs, the originator generates per-recipient information that either (a) encrypts the CEK for a particular recipient (ktri RecipientInfo CHOICE) or (b) transfers sufficient parameters to enable a particular recipient to independently generate the same KEK (kari RecipientInfo CHOICE). If the group is large, processing of the per-recipient information may take quite some time, not to mention the time required to collect and validate the PKCs for each of the recipients. Each recipient identifies its per-recipient information and uses the private key associated with the public key of its PKC to decrypt the CEK and hence gain access to the encrypted content.

With symmetric algorithms, the origination process is slightly different. Instead of using PKCs, the originator uses a previously distributed secret key-encryption key (KEK) to encrypt the CEK (kekri RecipientInfo CHOICE). Only one copy of the encrypted CEK is required because all the recipients already have the shared KEK needed to decrypt the CEK and hence gain access to the encrypted content.

The techniques to protect the shared KEK are beyond the scope of this document. Only the members of the list and the key manager should have the KEK in order to maintain confidentiality. Access control to the information protected by the KEK is determined by the entity that encrypts the information, as all members of the group have access. If the entity performing the encryption wants to ensure that some subset of the group does not gain access to the information, either a different KEK should be used (shared only with this smaller group) or asymmetric algorithms should be used.

1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

1.2. Applicability to E-mail

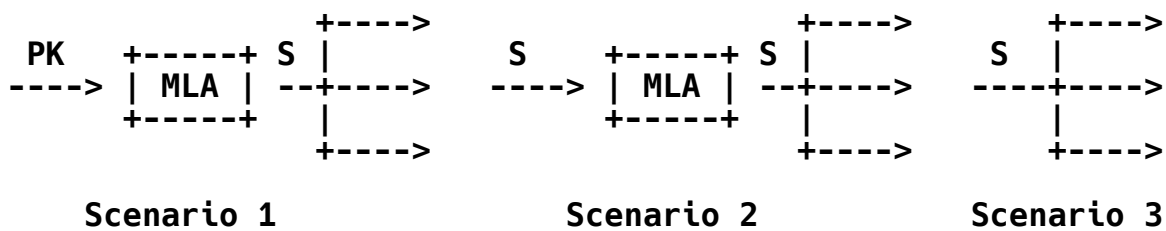
One primary audience for this distribution mechanism is e-mail. Distribution lists, sometimes referred to as mail lists, support the distribution of messages to recipients subscribed to the mail list. There are two models for how the mail list can be used. If the originator is a member of the mail list, the originator sends messages encrypted with the shared KEK to the mail list (e.g., listserv or majordomo) and the message is distributed to the mail list members. If the originator is not a member of the mail list (does not have the shared KEK), the originator sends the message (encrypted for the MLA) to the Mail List Agent (MLA), and then the MLA uses the shared KEK to encrypt the message for the members. In either case, the recipients of the mail list use the previously distributed-shared KEK to decrypt the message.

1.3. Applicability to Repositories

Objects can also be distributed via a repository (e.g., Lightweight Directory Access Protocol (LDAP) servers, X.500 Directory System Agents (DSAs), Web-based servers). If an object is stored in a repository encrypted with a symmetric key algorithm, anyone with the shared KEK and access to that object can then decrypt that object. The encrypted object and the encrypted, shared KEK can be stored in the repository.

1.4. Using the Group Key

This document was written with three specific scenarios in mind: two supporting Mail List Agents and one for general message distribution. Scenario 1 depicts the originator sending a public key (PK) protected message to an MLA who then uses the shared KEK(s) to redistribute the message to the members of the list. Scenario 2 depicts the originator sending a shared KEK protected message to an MLA who then redistributes the message to the members of the list (the MLA only adds additional recipients). The key used by the originator could be a key shared either amongst all recipients or just between the member and the MLA. Note that if the originator uses a key shared only with the MLA, then the MLA will need to decrypt the message and reencrypt the message for the list recipients. Scenario 3 shows an originator sending a shared KEK protected message to a group of recipients without an intermediate MLA.



2. Architecture

Figure 1 depicts the architecture to support symmetric key distribution. The Group List Agent (GLA) supports two distinct functions with two different agents:

- The Key Management Agent (KMA), which is responsible for generating the shared KEKs.
- The Group Management Agent (GMA), which is responsible for managing the Group List (GL) to which the shared KEKs are distributed.

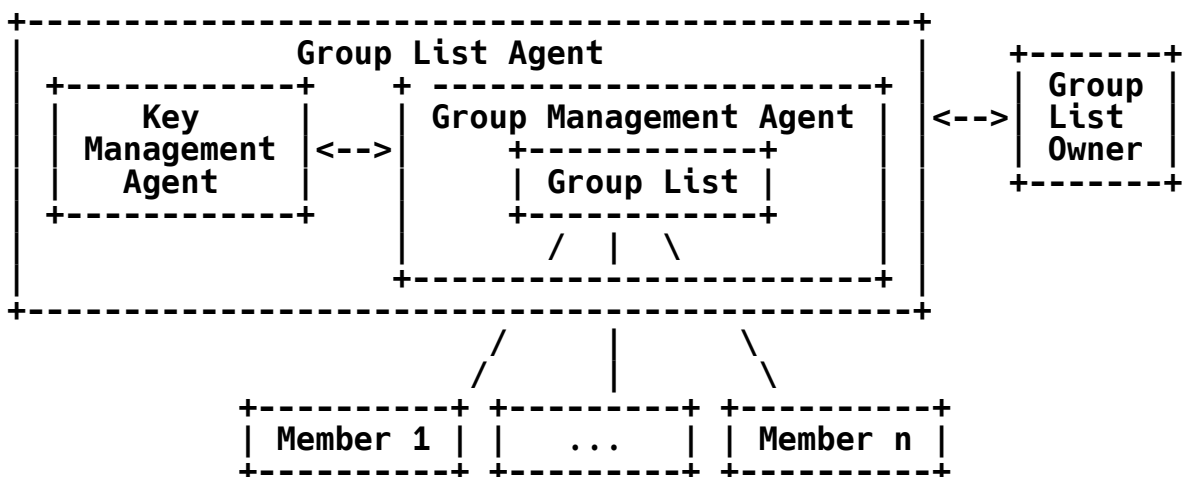


Figure 1 - Key Distribution Architecture

A GLA may support multiple KMAs. A GLA in general supports only one GMA, but the GMA may support multiple GLs. Multiple KMAs may support a GMA in the same fashion as GLAs support multiple KMAs. Assigning a particular KMA to a GL is beyond the scope of this document.

Modeling real-world GL implementations shows that there are very restrictive GLs, where a human determines GL membership, and very open GLs, where there are no restrictions on GL membership. To support this spectrum, the mechanism described herein supports both

managed (i.e., where access control is applied) and unmanaged (i.e., where no access control is applied) GLs. The access control mechanism for managed lists is beyond the scope of this document. Note: If the distribution for the list is performed by an entity other than the originator (e.g., an MLA distributing a mail message), this entity can also enforce access control rules.

In either case, the GL must initially be constructed by an entity hereafter called the Group List Owner (GLO). There may be multiple entities who 'own' the GL and who are allowed to make changes to the GL's properties or membership. The GLO determines if the GL will be managed or unmanaged and is the only entity that may delete the GL. GLO(s) may or may not be GL members. GLO(s) may also set up lists that are closed, where the GLO solely determines GL membership.

Though Figure 1 depicts the GLA as encompassing both the KMA and GMA functions, the two functions could be supported by the same entity or they could be supported by two different entities. If two entities are used, they could be located on one or two platforms. There is however a close relationship between the KMA and GMA functions. If the GMA stores all information pertaining to the GLs and the KMA merely generates keys, a corrupted GMA could cause havoc. To protect against a corrupted GMA, the KMA would be forced to double check the requests it receives to ensure that the GMA did not tamper with them. These duplicative checks blur the functionality of the two components together. For this reason, the interactions between the KMA and GMA are beyond the scope of this document.

Proprietary mechanisms may be used to separate the functions by strengthening the trust relationship between the two entities. Henceforth, the distinction between the two agents is not discussed further; the term GLA will be used to address both functions. It should be noted that a corrupt GLA can always cause havoc.

3. Protocol Interactions

There are existing mechanisms (e.g., listserv and majordomo) to manage GLs; however, this document does not address securing these mechanisms, as they are not standardized. Instead, it defines protocol interactions, as depicted in Figure 2, used by the GL members, GLA, and GLO(s) to manage GLs and distribute shared KEKs. The interactions have been divided into administration messages and distribution messages. The administrative messages are the request and response messages needed to set up the GL, delete the GL, add members to the GL, delete members of the GL, request a group rekey, add owners to the GL, remove owners of the GL, indicate a group key compromise, refresh a group key, interrogate the GLA, and update members' and owners' public key certificates. The distribution

messages are the messages that distribute the shared KEKs. The following sections describe the ASN.1 for both the administration and distribution messages. Section 4 describes how to use the administration messages, and Section 5 describes how to use the distribution messages.

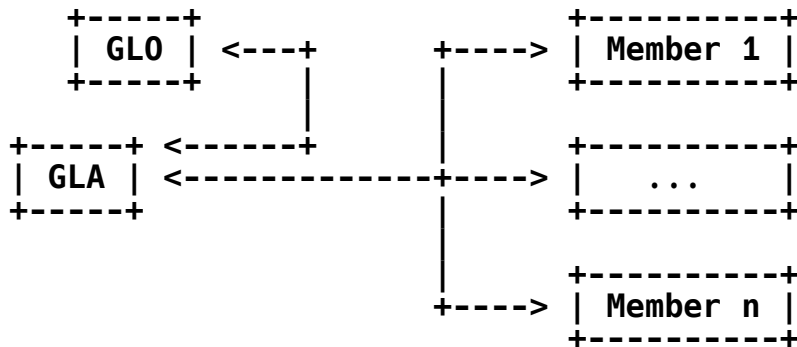


Figure 2 - Protocol Interactions

3.1. Control Attributes

To avoid creating an entirely new protocol, the Certificate Management over CMS (CMC) protocol was chosen as the foundation of this protocol. The main reason for the choice was the layering aspect provided by CMC where one or more control attributes are included in message, protected with CMS, to request or respond to a desired action. The CMC PKIData structure is used for requests, and the CMC PKIResponse structure is used for responses. The content-types PKIData and PKIResponse are then encapsulated in CMS's SignedData or EnvelopedData, or a combination of the two (see Section 3.2). The following are the control attributes defined in this document:

Control Attribute	OID	Syntax
glUseKEK	id-skd 1	GLUseKEK
glDelete	id-skd 2	GeneralName
glAddMember	id-skd 3	GLAddMember
glDeleteMember	id-skd 4	GLDeleteMember
glRekey	id-skd 5	GLRekey
glAddOwner	id-skd 6	GLOwnerAdministration
glRemoveOwner	id-skd 7	GLOwnerAdministration
glkCompromise	id-skd 8	GeneralName
glkRefresh	id-skd 9	GLKRefresh
glQueryRequest	id-skd 11	GLAQueryRequest
glQueryResponse	id-skd 12	GLAQueryResponse
glProvideCert	id-skd 13	GLManageCert
glUpdateCert	id-skd 14	GLManageCert
glKey	id-skd 15	GLKey

In the following conformance tables, the column headings have the following meanings: O for originate, R for receive, and F for forward. There are three types of implementations: GLOs, GLAs, and GL members. The GLO is an optional component, hence all GLO O and GLO R messages are optional, and GLA F messages are optional. The first table includes messages that conformant implementations **MUST** support. The second table includes messages that **MAY** be implemented. The second table should be interpreted as follows: if the control attribute is implemented by a component, then it must be implemented as indicated. For example, if a GLA is implemented that supports the glAddMember control attribute, then it **MUST** support receiving the glAddMember message. Note that "-" means not applicable.

Implementation			Required Requirement			Control Attribute	
GLO		GLA		GL Member			
O	R	O	R	F	O		R
-----		-----			-----		
MAY	-	MUST	-	MAY	-	MUST	glProvideCert
MAY	MAY	-	MUST	MAY	MUST	-	glUpdateCert
-	-	MUST	-	-	-	MUST	glKey

GLO		Implementation Requirement			Optional GL Member		Control Attribute
0	R	0	GLA R	F	0	R	
MAY	-	-	MAY	-	-	-	glUseKEK
MAY	-	-	MAY	-	-	-	glDelete
MAY	MAY	-	MUST	MAY	MUST	-	glAddMember
MAY	MAY	-	MUST	MAY	MUST	-	glDeleteMember
MAY	-	-	MAY	-	-	-	glRekey
MAY	-	-	MAY	-	-	-	glAddOwner
MAY	-	-	MAY	-	-	-	glRemoveOwner
MAY	MAY	-	MUST	MAY	MUST	-	glkCompromise
MAY	-	-	MUST	-	MUST	-	glkRefresh
MAY	-	-	SHOULD	-	MAY	-	glaQueryRequest
-	MAY	SHOULD	-	-	-	MAY	glaQueryResponse

glaQueryResponse is carried in the CMC PKIResponse content-type, all other control attributes are carried in the CMC PKIData content-type. The exception is glUpdateCert, which can be carried in either PKIData or PKIResponse.

Success and failure messages use CMC (see Section 3.2.4).

3.1.1. GL Use KEK

The GLO uses glUseKEK to request that a shared KEK be assigned to a GL. glUseKEK messages MUST be signed by the GLO. The glUseKEK control attribute has the syntax GLUseKEK:

```
GLUseKEK ::= SEQUENCE {
    glInfo          GLInfo,
    glOwnerInfo     SEQUENCE SIZE (1..MAX) OF GLOwnerInfo,
    glAdministration GLAdministration DEFAULT 1,
    glKeyAttributes GLKeyAttributes OPTIONAL }
```

```
GLInfo ::= SEQUENCE {
    glName      GeneralName,
    glAddress   GeneralName }
```

```
GLOwnerInfo ::= SEQUENCE {
    glOwnerName      GeneralName,
    glOwnerAddress   GeneralName,
    certificate       Certificates OPTIONAL }
```

```

Certificates ::= SEQUENCE {
    pKC          [0] Certificate OPTIONAL,
                -- See [PROFILE]
    aC           [1] SEQUENCE SIZE (1.. MAX) OF
                AttributeCertificate OPTIONAL,
                -- See [ACPROF]
    certPath     [2] CertificateSet OPTIONAL }
                -- From [CMS]

-- CertificateSet and CertificateChoices are included only
-- for illustrative purposes as they are imported from [CMS].

CertificateSet ::= SET SIZE (1..MAX) OF CertificateChoices

-- CertificateChoices supports X.509 public key certificates in
-- certificates and v2 attribute certificates in v2AttrCert.

GLAdministration ::= INTEGER {
    unmanaged    (0),
    managed      (1),
    closed       (2) }

GLKeyAttributes ::= SEQUENCE {
    rekeyControlledByGLO [0] BOOLEAN DEFAULT FALSE,
    recipientsNotMutuallyAware [1] BOOLEAN DEFAULT TRUE,
    duration             [2] INTEGER DEFAULT 0,
    generationCounter    [3] INTEGER DEFAULT 2,
    requestedAlgorithm    [4] AlgorithmIdentifier
                        DEFAULT { id-aes128-wrap } }

```

The fields in GLUseKEK have the following meaning:

- glInfo indicates the name of the GL in glName and the address of the GL in glAddress. The glName and glAddress can be the same, but this is not always the case. Both the name and address MUST be unique for a given GLA.
- glOwnerInfo indicates:
 - glOwnerName indicates the name of the owner of the GL. One of the names in glOwnerName MUST match one of the names in the certificate (either the subject distinguished name or one of the subject alternative names) used to sign this SignedData.PKIData creating the GL (i.e., the immediate signer).
 - glOwnerAddress indicates the GL owner's address.

- certificates MAY be included. It contains the following three fields:
 - certificates.pKC includes the encryption certificate for the GLO. It will be used to encrypt responses for the GLO.
 - certificates.aC MAY be included to convey any attribute certificate (see [ACPROF]) associated with the encryption certificate of the GLO included in certificates.pKC.
 - certificates.certPath MAY also be included to convey certificates that might aid the recipient in constructing valid certification paths for the certificate provided in certificates.pKC and the attribute certificates provided in certificates.aC. These certificates are optional because they might already be included elsewhere in the message (e.g., in the outer CMS layer).
- glAdministration indicates how the GL ought to be administered. The default is for the list to be managed. Three values are supported for glAdministration:
 - Unmanaged - When the GLO sets glAdministration to unmanaged, it is allowing prospective members to request addition and deletion from the GL without GLO intervention.
 - Managed - When the GLO sets glAdministration to managed, it is allowing prospective members to request addition and deletion from the GL, but the request is redirected by the GLA to GLO for review. The GLO makes the determination as to whether to honor the request.
 - Closed - When the GLO sets glAdministration to closed, it is not allowing prospective members to request addition or deletion from the GL. The GLA will only accept glAddMember and glDeleteMember requests from the GLO.
- glKeyAttributes indicates the attributes the GLO wants the GLA to assign to the shared KEK. If this field is omitted, GL rekeys will be controlled by the GLA, the recipients are allowed to know about one another, the algorithm will be AES-128 (see Section 7), the shared KEK will be valid for a calendar month (i.e., first of the month until the last day

of the month), and two shared KEKs will be distributed initially. The fields in `glKeyAttributes` have the following meaning:

- `rekeyControlledByGLO` indicates whether the GL rekey messages will be generated by the GLO or by the GLA. The default is for the GLA to control rekeys. If GL rekey is controlled by the GLA, the GL will continue to be rekeyed until the GLO deletes the GL or changes the GL rekey to be GLO controlled.
- `recipientsNotMutuallyAware` indicates that the GLO wants the GLA to distribute the shared KEK individually for each of the GL members (i.e., a separate `glKey` message is sent to each recipient). The default is for separate `glKey` message not to be required.

Note: This supports lists where one member does not know the identities of the other members. For example, a list is configured granting submit permissions to only one member. All other members are 'listening'. The security policy of the list does not allow the members to know who else is on the list. If a `glKey` is constructed for all of the GL members, information about each of the members may be derived from the information in `RecipientInfos`.

To make sure the `glkey` message does not divulge information about the other recipients, a separate `glKey` message would be sent to each GL member.

- `duration` indicates the length of time (in days) during which the shared KEK is considered valid. The value zero (0) indicates that the shared KEK is valid for a calendar month in the UTC Zulu time zone. For example, if the duration is zero (0), if the GL shared KEK is requested on July 24, the first key will be valid until the end of July and the next key will be valid for the entire month of August. If the value is not zero (0), the shared KEK will be valid for the number of days indicated by the value. For example, if the value of duration is seven (7) and the shared KEK is requested on Monday but not generated until Tuesday (13 May 2008); the shared KEKs will be valid from Tuesday (13 May 2008) to Tuesday (20 May 2008). The exact time of the day is determined when the key is generated.

- generationCounter indicates the number of keys the GLO wants the GLA to distribute. To ensure uninterrupted function of the GL, two (2) shared KEKs at a minimum MUST be initially distributed. The second shared KEK is distributed with the first shared KEK, so that when the first shared KEK is no longer valid the second key can be used. If the GLA controls rekey, then it also indicates the number of shared KEKs the GLO wants outstanding at any one time. See Sections 4.5 and 5 for more on rekey.
- requestedAlgorithm indicates the algorithm and any parameters the GLO wants the GLA to use with the shared KEK. The parameters are conveyed via the SMIMECapabilities attribute (see [MSG]). See Section 6 for more on algorithms.

3.1.2. Delete GL

GLOs use glDelete to request that a GL be deleted from the GLA. The glDelete control attribute has the syntax GeneralName. The glDelete message MUST be signed by the GLO. The name of the GL to be deleted is included in GeneralName:

DeleteGL ::= GeneralName

3.1.3. Add GL Member

GLOs use the glAddMember to request addition of new members, and prospective GL members use the glAddMember to request their own addition to the GL. The glAddMember message MUST be signed by either the GLO or the prospective GL member. The glAddMember control attribute has the syntax GLAddMember:

GLAddMember ::= SEQUENCE {
 glName GeneralName,
 glMember GLMember }

GLMember ::= SEQUENCE {
 glMemberName GeneralName,
 glMemberAddress GeneralName OPTIONAL,
 certificates Certificates OPTIONAL }

The fields in GLAddMembers have the following meaning:

- glName indicates the name of the GL to which the member should be added.

- glMember indicates the particulars for the GL member. Both of the following fields must be unique for a given GL:
 - glMemberName indicates the name of the GL member.
 - glMemberAddress indicates the GL member's address. It MUST be included.
- Note: In some instances, the glMemberName and glMemberAddress may be the same, but this is not always the case.
- certificates MUST be included. It contains the following three fields:
 - certificates.pKC includes the member's encryption certificate. It will be used, at least initially, to encrypt the shared KEK for that member. If the message is generated by a prospective GL member, the pKC MUST be included. If the message is generated by a GLO, the pKC SHOULD be included.
 - certificates.aC MAY be included to convey any attribute certificate (see [ACPROF]) associated with the member's encryption certificate.
 - certificates.certPath MAY also be included to convey certificates that might aid the recipient in constructing valid certification paths for the certificate provided in certificates.pKC and the attribute certificates provided in certificates.aC. These certificates are optional because they might already be included elsewhere in the message (e.g., in the outer CMS layer).

3.1.4. Delete GL Member

GLOs use the glDeleteMember to request deletion of GL members, and GL members use the glDeleteMember to request their own removal from the GL. The glDeleteMember message MUST be signed by either the GLO or the GL member. The glDeleteMember control attribute has the syntax GLDeleteMember:

```
GLDeleteMember ::= SEQUENCE {  
    glName           GeneralName,  
    glMemberToDelete GeneralName }
```

The fields in GLDeleteMembers have the following meaning:

- glName indicates the name of the GL from which the member should be removed.
- glMemberToDelete indicates the name or address of the member to be deleted.

3.1.5. Rekey GL

GLOs use the glRekey to request a GL rekey. The glRekey message MUST be signed by the GLO. The glRekey control attribute has the syntax GLRekey:

```
GLRekey ::= SEQUENCE {  
    glName             GeneralName,  
    glAdministration   GLAdministration OPTIONAL,  
    glNewKeyAttributes GLNewKeyAttributes OPTIONAL,  
    glRekeyAllGLKeys   BOOLEAN OPTIONAL }
```

```
GLNewKeyAttributes ::= SEQUENCE {  
    rekeyControlledByGLO [0] BOOLEAN OPTIONAL,  
    recipientsNotMutuallyAware [1] BOOLEAN OPTIONAL,  
    duration [2] INTEGER OPTIONAL,  
    generationCounter [3] INTEGER OPTIONAL,  
    requestedAlgorithm [4] AlgorithmIdentifier OPTIONAL }
```

The fields in GLRekey have the following meaning:

- glName indicates the name of the GL to be rekeyed.
- glAdministration indicates if there is any change to how the GL should be administered. See Section 3.1.1 for the three options. This field is only included if there is a change from the previously registered glAdministration.
- glNewKeyAttributes indicates whether the rekey of the GLO is controlled by the GLA or GL, what algorithm and parameters the GLO wishes to use, the duration of the key, and how many keys will be issued. The field is only included if there is a change from the previously registered glKeyAttributes.
- glRekeyAllGLKeys indicates whether the GLO wants all of the outstanding GL's shared KEKs rekeyed. If it is set to TRUE then all outstanding KEKs MUST be issued. If it is set to FALSE then all outstanding KEKs need not be reissued.

3.1.6. Add GL Owner

GLOs use the `glAddOwner` to request that a new GLO be allowed to administer the GL. The `glAddOwner` message MUST be signed by a registered GLO. The `glAddOwner` control attribute has the syntax `GLOwnerAdministration`:

```
GLOwnerAdministration ::= SEQUENCE {  
    glName      GeneralName,  
    glOwnerInfo GLOwnerInfo }
```

The fields in `GLAddOwners` have the following meaning:

- `glName` indicates the name of the GL to which the new GLO should be associated.
- `glOwnerInfo` indicates the name, address, and certificates of the new GLO. As this message includes names of new GLOs, the `certificates.pKC` MUST be included, and it MUST include the encryption certificate of the new GLO.

3.1.7. Remove GL Owner

GLOs use the `glRemoveOwner` to request that a GLO be disassociated with the GL. The `glRemoveOwner` message MUST be signed by a registered GLO. The `glRemoveOwner` control attribute has the syntax `GLOwnerAdministration`:

```
GLOwnerAdministration ::= SEQUENCE {  
    glName      GeneralName,  
    glOwnerInfo GLOwnerInfo }
```

The fields in `GLRemoveOwners` have the following meaning:

- `glName` indicates the name of the GL to which the GLO should be disassociated.
- `glOwnerInfo` indicates the name and address of the GLO to be removed. The `certificates` field SHOULD be omitted, as it will be ignored.

3.1.8. GL Key Compromise

GL members and GLOs use `glkCompromise` to indicate that the shared KEK possessed has been compromised. The `glKeyCompromise` control attribute has the syntax `GeneralName`. This message is always redirected by the GLA to the GLO for further action. The `glkCompromise` MAY be included in an `EnvelopedData` generated with the

compromised shared KEK. The name of the GL to which the compromised key is associated is placed in GeneralName:

GLKCompromise ::= GeneralName

3.1.9. GL Key Refresh

GL members use the glkRefresh to request that the shared KEK be redistributed to them. The glkRefresh control attribute has the syntax GLKRefresh.

GLKRefresh ::= SEQUENCE {
 glName GeneralName,
 dates SEQUENCE SIZE (1..MAX) OF Date }

Date ::= SEQUENCE {
 start GeneralizedTime,
 end GeneralizedTime OPTIONAL }

The fields in GLKRefresh have the following meaning:

- glName indicates the name of the GL for which the GL member wants shared KEKs.
- dates indicates a date range for keys the GL member wants. The start field indicates the first date the GL member wants and the end field indicates the last date. The end date MAY be omitted to indicate the GL member wants all keys from the specified start date to the current date. Note that a procedural mechanism is needed to restrict users from accessing messages that they are not allowed to access.

3.1.10. GLA Query Request and Response

There are situations where GLOs and GL members may need to determine some information from the GLA about the GL. GLOs and GL members use the glaQueryRequest, defined in Section 3.1.10.1, to request information and GLAs use the glaQueryResponse, defined in Section 3.1.10.2, to return the requested information. Section 3.1.10.3 includes one request and response type and value; others may be defined in additional documents.

3.1.10.1. GLA Query Request

GLOs and GL members use the glaQueryRequest to ascertain information about the GLA. The glaQueryRequest control attribute has the syntax GLAQueryRequest:

```
GLAQueryRequest ::= SEQUENCE {  
    glaRequestType OBJECT IDENTIFIER,  
    glaRequestValue ANY DEFINED BY glaRequestType }
```

3.1.10.2. GLA Query Response

GLAs return the glaQueryResponse after receiving a GLAQueryRequest. The glaQueryResponse MUST be signed by a GLA. The glaQueryResponse control attribute has the syntax GLAQueryResponse:

```
GLAQueryResponse ::= SEQUENCE {  
    glaResponseType OBJECT IDENTIFIER,  
    glaResponseValue ANY DEFINED BY glaResponseType }
```

3.1.10.3. Request and Response Types

Requests and responses are registered as a pair under the following object identifier arc:

```
id-cmc-glaRR OBJECT IDENTIFIER ::= { id-cmc 99 }
```

This document defines one request/response pair for GL members and GLOs to query the GLA for the list of algorithm it supports. The following Object Identifier (OID) is included in the glaQueryType field:

```
id-cmc-gla-skdAlgRequest OBJECT IDENTIFIER ::= { id-cmc-glaRR 1 }
```

```
SKDAlgRequest ::= NULL
```

If the GLA supports GLAQueryRequest and GLAQueryResponse messages, the GLA may return the following OID in the glaQueryType field:

```
id-cmc-gla-skdAlgResponse OBJECT IDENTIFIER ::= { id-cmc-glaRR 2 }
```

The glaQueryValue has the form of the smimeCapabilities attributes as defined in [MSG].

3.1.11. Provide Cert

GLAs and GLOs use the glProvideCert to request that a GL member provide an updated or new encryption certificate. The glProvideCert message MUST be signed by either GLA or GLO. If the GL member's PKC has been revoked, the GLO or GLA MUST NOT use it to generate the EnvelopedData that encapsulates the glProvideCert request. The glProvideCert control attribute has the syntax GLManageCert:

```
GLManageCert ::= SEQUENCE {  
    glName      GeneralName,  
    glMember    GLMember }
```

The fields in GLManageCert have the following meaning:

- glName indicates the name of the GL to which the GL member's new certificate is to be associated.
- glMember indicates particulars for the GL member:
 - glMemberName indicates the GL member's name.
 - glMemberAddress indicates the GL member's address. It MAY be omitted.
 - certificates SHOULD be omitted.

3.1.12 Update Cert

GL members and GLOs use the glUpdateCert to provide a new certificate for the GL. GL members can generate an unsolicited glUpdateCert or generate a response glUpdateCert as a result of receiving a glProvideCert message. GL members MUST sign the glUpdateCert. If the GL member's encryption certificate has been revoked, the GL member MUST NOT use it to generate the EnvelopedData that encapsulates the glUpdateCert request or response. The glUpdateCert control attribute has the syntax GLManageCert:

```
GLManageCert ::= SEQUENCE {  
    glName      GeneralName,  
    glMember    GLMember }
```

The fields in GLManageCert have the following meaning:

- glName indicates the name of the GL to which the GL member's new certificate should be associated.
- glMember indicates the particulars for the GL member:
 - glMemberName indicates the GL member's name.
 - glMemberAddress indicates the GL member's address. It MAY be omitted.
 - certificates MAY be omitted if the GLManageCert message is sent to request the GL member's certificate; otherwise, it MUST be included. It includes the following three fields:

- certificates.pKC includes the member's encryption certificate that will be used to encrypt the shared KEK for that member.
- certificates.aC MAY be included to convey one or more attribute certificates associated with the member's encryption certificate.
- certificates.certPath MAY also be included to convey certificates that might aid the recipient in constructing valid certification paths for the certificate provided in certificates.pKC and the attribute certificates provided in certificates.aC. These certificates are optional because they might already be included elsewhere in the message (e.g., in the outer CMS layer).

3.1.13. GL Key

The GLA uses the glKey to distribute the shared KEK. The glKey message MUST be signed by the GLA. The glKey control attribute has the syntax GLKey:

```
GLKey ::= SEQUENCE {  
    glName          GeneralName,  
    glIdentifier     KEKIdentifier,      -- See [CMS]  
    glkWrapped       RecipientInfos,     -- See [CMS]  
    glkAlgorithm     AlgorithmIdentifier,  
    glkNotBefore     GeneralizedTime,  
    glkNotAfter      GeneralizedTime }
```

-- KEKIdentifier is included only for illustrative purposes as
-- it is imported from [CMS].

```
KEKIdentifier ::= SEQUENCE {  
    keyIdentifier OCTET STRING,  
    date GeneralizedTime OPTIONAL,  
    other OtherKeyAttribute OPTIONAL }
```

The fields in GLKey have the following meaning:

- glName is the name of the GL.
- glIdentifier is the key identifier of the shared KEK. See Section 6.2.3 of [CMS] for a description of the subfields.

- `glkWrapped` is the wrapped shared KEK for the GL for a particular duration. The `RecipientInfos` MUST be generated as specified in Section 6.2 of [CMS]. The `ktri RecipientInfo` choice MUST be supported. The key in the `EncryptedKey` field (i.e., the distributed shared KEK) MUST be generated according to the section concerning random number generation in the security considerations of [CMS].
- `glkAlgorithm` identifies the algorithm with which the shared KEK is used. Since no encrypted data content is being conveyed at this point, the parameters encoded with the algorithm should be the structure defined for `smimeCapabilities` rather than encrypted content.
- `glkNotBefore` indicates the date at which the shared KEK is considered valid. `GeneralizedTime` values MUST be expressed in UTC (Zulu) and MUST include seconds (i.e., times are `YYYYMMDDHHMMSSZ`), even where the number of seconds is zero. `GeneralizedTime` values MUST NOT include fractional seconds.
- `glkNotAfter` indicates the date after which the shared KEK is considered invalid. `GeneralizedTime` values MUST be expressed in UTC (Zulu) and MUST include seconds (i.e., times are `YYYYMMDDHHMMSSZ`), even where the number of seconds is zero. `GeneralizedTime` values MUST NOT include fractional seconds.

If the `glKey` message is in response to a `glUseKEK` message:

- The GLA MUST generate separate `glKey` messages for each recipient if `glUseKEK.glKeyAttributes.recipientsNotMutuallyAware` is set to `TRUE`. For each recipient, you want to generate a message that contains that recipient's key (i.e., one message with one attribute).
- The GLA MUST generate the requested number of `glKey` messages. The value in `glUseKEK.glKeyAttributes.generationCounter` indicates the number of `glKey` messages requested.

If the `glKey` message is in response to a `glRekey` message:

- The GLA MUST generate separate `glKey` messages for each recipient if `glRekey.glNewKeyAttributes.recipientsNotMutuallyAware` is set to `TRUE`.
- The GLA MUST generate the requested number of `glKey` messages. The value in `glUseKEK.glKeyAttributes.generationCounter` indicates the number of `glKey` messages requested.

- The GLA **MUST** generate one glKey message for each outstanding shared KEKs for the GL when glRekeyAllGLKeys is set to TRUE.

If the glKey message was not in response to a glRekey or glUseKEK (e.g., where the GLA controls rekey):

- The GLA **MUST** generate separate glKey messages for each recipient when glUseKEK.glNewKeyAttributes.recipientsNotMutuallyAware that set up the GL was set to TRUE.
- The GLA **MAY** generate glKey messages prior to the duration on the last outstanding shared KEK expiring, where the number of glKey messages generated is generationCounter minus one (1). Other distribution mechanisms can also be supported to support this functionality.

3.2. Use of CMC, CMS, and PKIX

The following sections outline the use of CMC, CMS, and the PKIX certificate and CRL profile.

3.2.1. Protection Layers

The following sections outline the protection required for the control attributes defined in this document.

Note: There are multiple ways to encapsulate SignedData and EnvelopedData. The first is to use a MIME wrapper around each ContentInfo, as specified in [MSG]. The second is not to use a MIME wrapper around each ContentInfo, as specified in Transporting S/MIME Objects in X.400 [X400TRANS].

3.2.1.1. Minimum Protection

At a minimum, a SignedData **MUST** protect each request and response encapsulated in PKIData and PKIResponse. The following is a depiction of the minimum wrappings:

Minimum Protection

SignedData

PKIData or PKIResponse
controlSequence

Prior to taking any action on any request or response SignedData(s) **MUST** be processed according to [CMS].

3.2.1.2. Additional Protection

An additional EnvelopedData MAY also be used to provide confidentiality of the request and response. An additional SignedData MAY also be added to provide authentication and integrity of the encapsulated EnvelopedData. The following is a depiction of the optional additional wrappings:

Confidentiality Protection -----	Authentication and Integrity of Confidentiality Protection -----
EnvelopedData	SignedData
SignedData	EnvelopedData
PKIData or PKIResponse	SignedData
controlSequence	PKIData or PKIResponse
	controlSequence

If an incoming message is encrypted, the confidentiality of the message MUST be preserved. All EnvelopedData objects MUST be processed as specified in [CMS]. If a SignedData is added over an EnvelopedData, a ContentHints attribute SHOULD be added. See Section 2.9 of Extended Security Services for S/MIME [ESS].

If the GLO or GL member applies confidentiality to a request, the EnvelopedData MUST include the GLA as a recipient. If the GLA forwards the GL member request to the GLO, then the GLA MUST decrypt the EnvelopedData content, strip the confidentiality layer, and apply its own confidentiality layer as an EnvelopedData with the GLO as a recipient.

3.2.2. Combining Requests and Responses

Multiple requests and responses corresponding to a GL MAY be included in one PKIData.controlSequence or PKIResponse.controlSequence. Requests and responses for multiple GLs MAY be combined in one PKIData or PKIResponse by using PKIData.cmsSequence and PKIResponse.cmsSequence. A separate cmsSequence MUST be used for different GLs. That is, requests corresponding to two different GLs are included in different cmsSequences. The following is a diagram depicting multiple requests and responses combined in one PKIData and PKIResponse:

Multiple Requests and Responses Request -----	Response -----
SignedData	SignedData
PKIData	PKIResponse
cmsSequence	cmsSequence
SignedData	SignedData
PKIData	PKIResponse
controlSequence	controlSequence
One or more requests	One or more responses
corresponding to one GL	corresponding to one GL
SignedData	SignedData
PKIData	PKIResponse
controlSequence	controlSequence
One or more requests	One or more responses
corresponding to another GL	corresponding to another GL

When applying confidentiality to multiple requests and responses, all of the requests/responses MAY be included in one EnvelopedData. The following is a depiction:

Confidentiality of Multiple Requests and Responses Wrapped Together

```

-----
EnvelopedData
  SignedData
    PKIData
    cmsSequence
    SignedData
    PKIResponse
    controlSequence
    One or more requests
    corresponding to one GL
  SignedData
    PKIData
    controlSequence
    One or more requests
    corresponding to one GL

```

Certain combinations of requests in one `PKIData.controlSequence` and one `PKIResponse.controlSequence` are not allowed. The invalid combinations listed here **MUST NOT** be generated:

Invalid Combinations

```
-----  
glUseKEK      & glDeleteMember  
glUseKEK      & glRekey  
glUseKEK      & glDelete  
glDelete      & glAddMember  
glDelete      & glDeleteMember  
glDelete      & glRekey  
glDelete      & glAddOwner  
glDelete      & glRemoveOwner
```

To avoid unnecessary errors, certain requests and responses **SHOULD** be processed prior to others. The following is the priority of message processing, if not listed it is an implementation decision as to which to process first: `glUseKEK` before `glAddMember`, `glRekey` before `glAddMember`, and `glDeleteMember` before `glRekey`. Note that there is a processing priority, but it does not imply an ordering within the content.

3.2.3. GLA Generated Messages

When the GLA generates a success or fail message, it generates one for each request. `SKDFailInfo` values of `unsupportedDuration`, `unsupportedDeliveryMethod`, `unsupportedAlgorithm`, `noGLONameMatch`, `nameAlreadyInUse`, `alreadyAnOwner`, and `notAnOwner` are not returned to GL members.

If `GLKeyAttributes.recipientsNotMutuallyAware` is set to `TRUE`, a separate `PKIResponse.cMCStatusInfoExt` and `PKIData.glKey` **MUST** be generated for each recipient. However, it is valid to send one message with multiple attributes to the same recipient.

If the GL has multiple GLOs, the GLA **MUST** send `cMCStatusInfoExt` messages to the requesting GLO. The mechanism to determine which GLO made the request is beyond the scope of this document.

If a GL is managed and the GLA receives a `glAddMember`, `glDeleteMember`, or `glCompromise` message, the GLA redirects the request to the GLO for review. An additional, `SignedData` **MUST** be applied to the redirected request as follows:

GLA Forwarded Requests

```

-----
SignedData
  PKIData
    cmsSequence
      SignedData
        PKIData
          controlSequence

```

3.2.4. CMC Control Attributes and CMS Signed Attributes

CMC carries control attributes as CMS signed attributes. These attributes are defined in [CMC] and [CMS]. Some of these attributes are REQUIRED; others are OPTIONAL. The required attributes are as follows: `cmCStatusInfoExt` `transactionId`, `senderNonce`, `recipientNonce`, `queryPending`, and `signingTime`. Other attributes can also be used; however, their use is beyond the scope of this document. The following sections specify requirements in addition to those already specified in [CMC] and [CMS].

3.2.4.1. Using `cmCStatusInfoExt`

`cmCStatusInfoExt` is used by GLAs to indicate to GLOs and GL members that a request was unsuccessful. Two classes of failure codes are used within this document. Errors from the `CMCFailInfo` list, found in Section 5.1.4 of CMC, are encoded as defined in CMC. Error codes defined in this document are encoded using the `ExtendedFailInfo` field of the `cmCStatusInfoExt` structure. If the same failure code applies to multiple commands, a single `cmCStatusInfoExt` structure can be used with multiple items in `cmCStatusInfoExt.bodyList`. The GLA MAY also return other pertinent information in `statusString`. The `SKDFailInfo` object identifier and value are:

```

id-cet-skdFailInfo OBJECT IDENTIFIER ::= { iso(1)
  identified-organization(3) dod(6) internet(1) security(5)
  mechanisms(5) pkix(7) cet(15) skdFailInfo(1) }

```

```

SKDFailInfo ::= INTEGER {
  unspecified          (0),
  closedGL             (1),
  unsupportedDuration   (2),
  noGLACertificate      (3),
  invalidCert          (4),
  unsupportedAlgorithm  (5),
  noGLONameMatch       (6),
  invalidGLName        (7),
  nameAlreadyInUse     (8),
  noSpam               (9),

```

```
-- obsolete          (10),  
  alreadyAMember      (11),  
  notAMember          (12),  
  alreadyAnOwner      (13),  
  notAnOwner          (14) }
```

The values have the following meaning:

- unspecified indicates that the GLA is unable or unwilling to perform the requested action and does not want to indicate the reason.
- closedGL indicates that members can only be added or deleted by the GLO.
- unsupportedDuration indicates that the GLA does not support generating keys that are valid for the requested duration.
- noGLACertificate indicates that the GLA does not have a valid certificate.
- invalidCert indicates that the member's encryption certificate was not verifiable (i.e., signature did not validate, certificate's serial number present on a CRL, the certificate expired, etc.).
- unsupportedAlgorithm indicates the GLA does not support the requested algorithm.
- noGLONameMatch indicates that one of the names in the certificate used to sign a request does not match the name of a registered GLO.
- invalidGLName indicates that the GLA does not support the glName present in the request.
- nameAlreadyInUse indicates that the glName is already assigned on the GLA.
- noSpam indicates that the prospective GL member did not sign the request (i.e., if the name in glMember.glMemberName does not match one of the names (either the subject distinguished name or one of the subject alternative names) in the certificate used to sign the request).
- alreadyAMember indicates that the prospective GL member is already a GL member.

- `notAMember` indicates that the prospective GL member to be deleted is not presently a GL member.
- `alreadyAnOwner` indicates that the prospective GLO is already a GLO.
- `notAnOwner` indicates that the prospective GLO to be deleted is not presently a GLO.

`CMCStatusInfoExt` is used by GLAs to indicate to GLOs and GL members that a request was successfully completed. If the request was successful, the GLA returns a `CMCStatusInfoExt` response with `CMCStatus.success` and optionally other pertinent information in `statusString`.

When the GL is managed and the GLO has reviewed GL member initiated `glAddMember`, `glDeleteMember`, and `glkComrpomise` requests, the GLO uses `CMCStatusInfoExt` to indicate the success or failure of the request. If the request is allowed, `CMCStatus.success` is returned and `statusString` is optionally returned to convey additional information. If the request is denied, `CMCStatus.failed` is returned and `statusString` is optionally returned to convey additional information. Additionally, the appropriate `SKDFailInfo` can be included in `CMCStatusInfoExt.extendedFailInfo`.

`CMCStatusInfoExt` is used by GLOs, GLAs, and GL members to indicate that signature verification failed. If the signature failed to verify over any control attribute except a `CMCStatusInfoExt`, a `CMCStatusInfoExt` control attribute MUST be returned indicating `CMCStatus.failed` and `otherInfo.failInfo.badMessageCheck`. If the signature over the outermost `PKIData` failed, the `bodyList` value is zero (0). If the signature over any other `PKIData` failed, the `bodyList` value is the `bodyPartId` value from the request or response. GLOs and GL members who receive `CMCStatusInfoExt` messages whose signatures are invalid SHOULD generate a new request to avoid `badMessageCheck` message loops.

`CMCStatusInfoExt` is also used by GLOs and GLAs to indicate that a request could not be performed immediately. If the request could not be processed immediately by the GLA or GLO, the `CMCStatusInfoExt` control attribute MUST be returned indicating `CMCStatus.pending` and `otherInfo.pendInfo`. When requests are redirected to the GLO for approval (for managed lists), the GLA MUST NOT return a `CMCStatusInfoExt` indicating query pending.

`CMCStatusInfoExt` is also used by GLAs to indicate that a `glaQueryRequest` is not supported. If the `glaQueryRequest` is not supported, the `CMCStatusInfoExt` control attribute **MUST** be returned indicating `CMCStatus.noSupport` and `statusString` is optionally returned to convey additional information.

`CMCStatusInfoExt` is also used by GL members, GLOs, and GLAs to indicate that the `signingTime` (see Section 3.2.4.3) is not close enough to the locally specified time. If the local time is not close enough to the time specified in `signingTime`, a `CMCStatus.failed` and `otherInfo.failInfo.badTime` **MAY** be returned.

3.2.4.2. Using `transactionId`

`transactionId` **MAY** be included by GLOs, GLAs, or GL members to identify a given transaction. All subsequent requests and responses related to the original request **MUST** include the same `transactionId` control attribute. If GL members include a `transactionId` and the request is redirected to the GLO, the GLA **MAY** include an additional `transactionId` in the outer `PKIData`. If the GLA included an additional `transactionId` in the outer `PKIData`, when the GLO generates a `CMCStatusInfoExt` response it generates one for the GLA with the GLA's `transactionId` and one for the GL member with the GL member's `transactionId`.

3.2.4.3. Using Nonces and `signingTime`

The use of nonces (see Section 5.6 of [CMC]) and an indication of when the message was signed (see Section 11.3 of [CMS]) can be used to provide application-level replay prevention.

To protect the GL, all messages **MUST** include the `signingTime` attribute. Message originators and recipients can then use the time provided in this attribute to determine whether they have previously received the message.

If the originating message includes a `senderNonce`, the response to the message **MUST** include the received `senderNonce` value as the `recipientNonce` and a new value as the `senderNonce` value in the response.

If a GLA aggregates multiple messages together or forwards a message to a GLO, the GLA **MAY** optionally generate a new nonce value and include that in the wrapping message. When the response comes back from the GLO, the GLA builds a response to the originator(s) of the message(s) and deals with each of the nonce values from the originating messages.

For these attributes, it is necessary to maintain state information on exchanges to compare one result to another. The time period for which this information is maintained is a local policy.

3.2.4.4. CMC and CMS Attribute Support Requirements

The following are the implementation requirements for CMC control attributes and CMS signed attributes for an implementation to be considered conformant to this specification:

Implementation Requirement							Attribute
GLO		GLA			GL Member		
O	R	O	R	F	O	R	
-----		-----			-----		-----
MUST	MUST	MUST	MUST	-	MUST	MUST	cmcStatusInfoExt
MAY	MAY	MUST	MUST	-	MAY	MAY	transactionId
MAY	MAY	MUST	MUST	-	MAY	MAY	senderNonce
MAY	MAY	MUST	MUST	-	MAY	MAY	recepientNonce
MUST	MUST	MUST	MUST	-	MUST	MUST	SKDFailInfo
MUST	MUST	MUST	MUST	-	MUST	MUST	signingTime

3.2.5. Resubmitted GL Member Messages

When the GL is managed, the GLA forwards the GL member requests to the GLO for GLO approval by creating a new request message containing the GL member request(s) as a cmsSequence item. If the GLO approves the request, it can either add a new layer of wrapping and send it back to the GLA or create a new message and send it to the GLA. (Note in this case there are now 3 layers of PKIData messages with appropriate signing layers.)

3.2.6. PKIX Certificate and CRL Profile

Signatures, certificates, and CRLs are verified according to the PKIX profile [PROFILE].

Name matching is performed according to the PKIX profile [PROFILE].

All distinguished name forms must follow the UTF8String convention noted in the PKIX profile [PROFILE].

A certificate per GL would be issued to the GLA.

GL policy may mandate that the GL member's address be included in the GL member's certificate.

4. Administrative Messages

There are a number of administrative messages that must be exchanged to manage a GL. The following sections describe each request and response message combination in detail. The procedures defined in this section are not prescriptive.

4.1. Assign KEK to GL

Prior to generating a group key, a GL needs to be set up and a shared KEK assigned to the GL. Figure 3 depicts the protocol interactions to set up and assign a shared KEK. Note that error messages are not depicted in Figure 3. Additionally, behavior for the optional `transactionId`, `senderNonce`, and `recipientNonce` CMC control attributes is not addressed in these procedures.

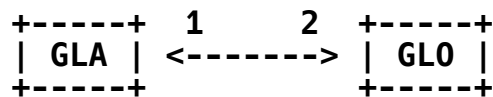


Figure 3 - Create Group List

The process is as follows:

- 1 - The GLO is the entity responsible for requesting the creation of the GL. The GLO sends a `SignedData.PKIData.controlSequence.glUseKEK` request to the GLA (1 in Figure 3). The GLO MUST include `glName`, `glAddress`, `glOwnerName`, `glOwnerAddress`, and `glAdministration`. The GLO MAY also include their preferences for the shared KEK in `glKeyAttributes` by indicating whether the GLO controls the rekey in `rekeyControlledByGLO`, whether separate `glKey` messages should be sent to each recipient in `recipientsNotMutuallyAware`, the requested algorithm to be used with the shared KEK in `requestedAlgorithm`, the duration of the shared KEK, and how many shared KEKs should be initially distributed in `generationCounter`. The GLO MUST also include the `signingTime` attribute with this request.
 - 1.a - If the GLO knows of members to be added to the GL, the `glAddMember` request(s) MAY be included in the same `controlSequence` as the `glUseKEK` request (see Section 3.2.2). The GLO indicates the same `glName` in the `glAddMember` request as in `glUseKEK.glInfo.glName`. Further `glAddMember` procedures are covered in Section 4.3.

- 1.b - The GLO can apply confidentiality to the request by encapsulating the SignedData.PKIData in an EnvelopedData (see Section 3.2.1.2).
- 1.c - The GLO can also optionally apply another SignedData over the EnvelopedData (see Section 3.2.1.2).
- 2 - Upon receipt of the request, the GLA checks the signingTime and verifies the signature on the innermost SignedData.PKIData. If an additional SignedData and/or EnvelopedData encapsulates the request (see Sections 3.2.1.2 and 3.2.2), the GLA verifies the outer signature(s) and/or decrypts the outer layer(s) prior to verifying the signature on the innermost SignedData.
 - 2.a - If the signingTime attribute value is not within the locally accepted time window, the GLA MAY return a response indicating cMCStatus.failed and otherInfo.failInfo.badTime and a signingTime attribute.
 - 2.b - Else if signature processing continues and if the signatures do not verify, the GLA returns a cMCStatusInfoExt response indicating cMCStatus.failed and otherInfo.failInfo.badMessageCheck. Additionally, a signingTime attribute is included with the response.
 - 2.c - Else if the signatures do verify but the GLA does not have a valid certificate, the GLA returns a cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of noValidGLACertificate. Additionally, a signingTime attribute is included with the response. Instead of immediately returning the error code, the GLA attempts to get a certificate, possibly using [CMC].
 - 2.d - Else the signatures are valid and the GLA does have a valid certificate, the GLA checks that one of the names in the certificate used to sign the request matches one of the names in glUseKEK.gLOwnerInfo.gLOwnerName.
 - 2.d.1 - If the names do not match, the GLA returns a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of noGLONameMatch. Additionally, a signingTime attribute is included with the response.

- 2.d.2 - Else if the names all match, the GLA checks that the `glName` and `glAddress` are not already in use. The GLA also checks any `glAddMember` included within the `controlSequence` with this `glUseKEK`. Further processing of the `glAddMember` is covered in Section 4.3.
- 2.d.2.a - If the `glName` is already in use, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `nameAlreadyInUse`. Additionally, a `signingTime` attribute is included with the response.
- 2.d.2.b - Else if the `requestedAlgorithm` is not supported, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `unsupportedAlgorithm`. Additionally, a `signingTime` attribute is included with the response.
- 2.d.2.c - Else if the duration cannot be supported, determining this is beyond the scope of this document, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `unsupportedDuration`. Additionally, a `signingTime` attribute is included with the response.
- 2.d.2.d - Else if the GL cannot be supported for other reasons, which the GLA does not wish to disclose, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `unspecified`. Additionally, a `signingTime` attribute is included with the response.
- 2.d.2.e - Else if the `glName` is not already in use, the duration can be supported, and the `requestedAlgorithm` is supported, the GLA MUST return a `CMCStatusInfoExt` indicating `CMCStatus.success` and a `signingTime` attribute. (2 in Figure 3). The GLA also takes administrative actions, which are beyond the scope of this document, to store the `glName`, `glAddress`, `glKeyAttributes`, `glOwnerName`, and `glOwnerAddress`. The GLA also sends a `glKey` message as described in section 5.

- 2.d.2.e.1 - The GLA can apply confidentiality to the response by encapsulating the SignedData.PKIResponse in an EnvelopedData if the request was encapsulated in an EnvelopedData (see Section 3.2.1.2).
 - 2.d.2.e.2 - The GLA can also optionally apply another SignedData over the EnvelopedData (see Section 3.2.1.2).
- 3 - Upon receipt of the cMCStatusInfoExt responses, the GLO checks the signingTime and verifies the GLA signature(s). If an additional SignedData and/or EnvelopedData encapsulates the response (see Section 3.2.1.2 or 3.2.2), the GLO verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost SignedData.
- 3.a - If the signingTime attribute value is not within the locally accepted time window, the GLO MAY return a response indicating cMCStatus.failed and otherInfo.failInfo.badTime and a signingTime attribute.
 - 3.b - Else if signature processing continues and if the signatures do verify, the GLO MUST check that one of the names in the certificate used to sign the response matches the name of the GL.
 - 3.b.1 - If the name of the GL does not match the name present in the certificate used to sign the message, the GLO should not believe the response.
 - 3.b.2 - Else if the name of the GL does match the name present in the certificate and:
 - 3.b.2.a - If the signatures do verify and the response was cMCStatusInfoExt indicating cMCStatus.success, the GLO has successfully created the GL.
 - 3.b.2.b - Else if the signatures are valid and the response is cMCStatusInfoExt.cMCStatus.failed with any reason, the GLO can reattempt to create the GL using the information provided in the response. The GLO can also use the glaQueryRequest to determine the algorithms and other characteristics supported by the GLA (see Section 4.9).

4.2. Delete GL from GLA

From time to time, there are instances when a GL is no longer needed. In this case, the GLO deletes the GL. Figure 4 depicts the protocol interactions to delete a GL. Note that behavior for the optional `transactionId`, `senderNonce`, and `recipientNonce` CMC control attributes is not addressed in these procedures.

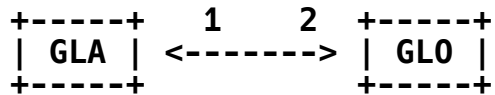


Figure 4 - Delete Group List

The process is as follows:

- 1 - The GLO is responsible for requesting the deletion of the GL. The GLO sends a `SignedData.PKIData.controlSequence.glDelete` request to the GLA (1 in Figure 4). The name of the GL to be deleted is included in `GeneralName`. The GLO MUST also include the `signingTime` attribute and can also include a `transactionId` and `senderNonce` attributes.
 - 1.a - The GLO can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` (see Section 3.2.1.2).
 - 1.b - The GLO MAY optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 2 - Upon receipt of the request, the GLA checks the `signingTime` and verifies the signature on the innermost `SignedData.PKIData`. If an additional `SignedData` and/or `EnvelopedData` encapsulates the request (see Section 3.2.1.2 or 3.2.2), the GLA verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 2.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLA MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
 - 2.b - Else if signature processing continues and if the signatures cannot be verified, the GLA returns a `CMCStatusInfoExt` response indicating `CMCStatus.failed` and `otherInfo.failInfo.badMessageCheck`. Additionally, a `signingTime` attribute is included with the response.

- 2.c - Else if the signatures verify, the GLA makes sure the GL is supported by checking the name of the GL matches a glName stored on the GLA.
 - 2.c.1 - If the glName is not supported by the GLA, the GLA returns a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of invalidGLName. Additionally, a signingTime attribute is included with the response.
 - 2.c.2 - Else if the glName is supported by the GLA, the GLA ensures that a registered GLO signed the glDelete request by checking if one of the names present in the digital signature certificate used to sign the glDelete request matches a registered GLO.
 - 2.c.2.a - If the names do not match, the GLA returns a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of noGLONameMatch. Additionally, a signingTime attribute is included with the response.
 - 2.c.2.b - Else if the names do match, but the GL cannot be deleted for other reasons, which the GLA does not wish to disclose, the GLA returns a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of unspecified. Additionally, a signingTime attribute is included with the response. Actions beyond the scope of this document must then be taken to delete the GL from the GLA.
 - 2.c.2.c - Else if the names do match, the GLA returns a cMCStatusInfoExt indicating cMCStatus.success and a signingTime attribute (2 in Figure 4). The GLA ought not accept further requests for member additions, member deletions, or group rekeys for this GL.
 - 2.c.2.c.1 - The GLA can apply confidentiality to the response by encapsulating the SignedData.PKIResponse in an EnvelopedData if the request was encapsulated in an EnvelopedData (see Section 3.2.1.2).
 - 2.c.2.c.2 - The GLA MAY optionally apply another SignedData over the EnvelopedData (see Section 3.2.1.2).

- 3 - Upon receipt of the `CMCStatusInfoExt` response, the GLO checks the `signingTime` and verifies the GLA signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the response (see Section 3.2.1.2 or 3.2.2), the GLO verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 3.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLO MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
 - 3.b - Else if signature processing continues and if the signatures verify, the GLO checks that one of the names in the certificate used to sign the response matches the name of the GL.
 - 3.b.1 - If the name of the GL does not match the name present in the certificate used to sign the message, the GLO should not believe the response.
 - 3.b.2 - Else if the name of the GL does match the name present in the certificate and:
 - 3.b.2.a - If the signatures verify and the response was `CMCStatusInfoExt` indicating `CMCStatus.success`, the GLO has successfully deleted the GL.
 - 3.b.2.b - Else if the signatures do verify and the response was `CMCStatusInfoExt.CMCStatus.failed` with any reason, the GLO can reattempt to delete the GL using the information provided in the response.

4.3. Add Members to GL

To add members to GLs, either the GLO or prospective members use the `glAddMember` request. The GLA processes GLO and prospective GL member requests differently though. GLOs can submit the request at any time to add members to the GL, and the GLA, once it has verified the request came from a registered GLO, should process it. If a prospective member sends the request, the GLA needs to determine how the GL is administered. When the GLO initially configured the GL, it set the GL to be unmanaged, managed, or closed (see Section 3.1.1). In the unmanaged case, the GLA merely processes the member's request. In the managed case, the GLA forwards the requests from the prospective members to the GLO for review. Where there are multiple GLOs for a GL, which GLO the request is forwarded to is beyond the scope of this document. The GLO reviews the request and either

rejects it or submits a reformed request to the GLA. In the closed case, the GLA will not accept requests from prospective members. The following sections describe the processing for the GLO(s), GLA, and prospective GL members depending on where the glAddMeber request originated, either from a GLO or from prospective members. Figure 5 depicts the protocol interactions for the three options. Note that the error messages are not depicted. Additionally, note that behavior for the optional transactionId, senderNonce, and recipientNonce CMC control attributes is not addressed in these procedures.

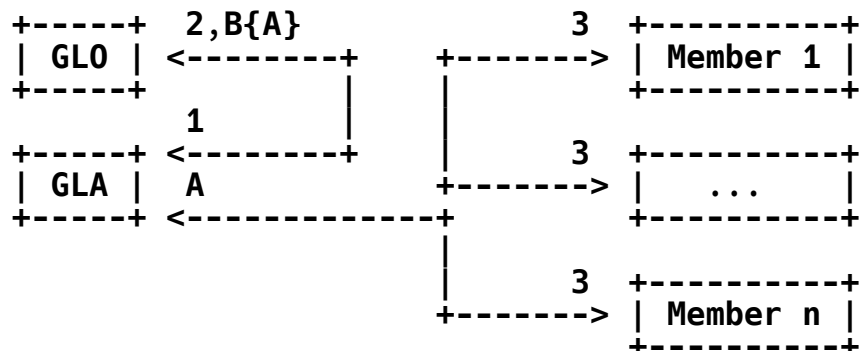


Figure 5 - Member Addition

An important decision that needs to be made on a group-by-group basis is whether to rekey the group every time a new member is added. Typically, unmanaged GLs should not be rekeyed when a new member is added, as the overhead associated with rekeying the group becomes prohibitive, as the group becomes large. However, managed and closed GLs can be rekeyed to maintain the confidentiality of the traffic sent by group members. An option to rekeying managed or closed GLs when a member is added is to generate a new GL with a different group key. Group rekeying is discussed in Sections 4.5 and 5.

4.3.1. GLO Initiated Additions

The process for GLO initiated glAddMember requests is as follows:

- 1 - The GLO collects the pertinent information for the member(s) to be added (this may be done through an out-of-bands means). The GLO then sends a SignedData.PKIData.controlSequence with a separate glAddMember request for each member to the GLA (1 in Figure 5). The GLO includes the GL name in glName, the member's name in glMember.glMemberName, the member's address in glMember.glMemberAddress, and the member's encryption certificate in glMember.certificates.pKC. The GLO can also include any attribute certificates associated with the member's encryption

certificate in `glMember.certificates.aC`, and the certification path associated with the member's encryption and attribute certificates in `glMember.certificates.certPath`. The GLO MUST also include the `signingTime` attribute with this request.

- 1.a - The GLO can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` (see Section 3.2.1.2).
- 1.b - The GLO can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 2 - Upon receipt of the request, the GLA checks the `signingTime` and verifies the signature on the innermost `SignedData.PKIData`. If an additional `SignedData` and/or `EnvelopedData` encapsulates the request (see Section 3.2.1.2 or 3.2.2), the GLA verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 2.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLA MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
 - 2.b - Else if signature processing continues and if the signatures cannot be verified, the GLA returns a `CMCStatusInfoExt` response indicating `CMCStatus.failed` and `otherInfo.failInfo.badMessageCheck`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c - Else if the signatures verify, the `glAddMember` request is included in a `controlSequence` with the `glUseKEK` request, and the processing in Section 4.1 item 2.d is successfully completed, the GLA returns a `CMCStatusInfoExt` indicating `CMCStatus.success` and a `signingTime` attribute (2 in Figure 5).
 - 2.c.1 - The GLA can apply confidentiality to the response by encapsulating the `SignedData.PKIData` in an `EnvelopedData` if the request was encapsulated in an `EnvelopedData` (see Section 3.2.1.2).
 - 2.c.2 - The GLA can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).

- 2.d - Else if the signatures verify and the GLAddMember request is not included in a controlSequence with the GLCreate request, the GLA makes sure the GL is supported by checking that the glName matches a glName stored on the GLA.
- 2.d.1 - If the glName is not supported by the GLA, the GLA returns a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of invalidGLName. Additionally, a signingTime attribute is included with the response.
- 2.d.2 - Else if the glName is supported by the GLA, the GLA checks to see if the glMemberName is present on the GL.
 - 2.d.2.a - If the glMemberName is present on the GL, the GLA returns a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of alreadyAMember. Additionally, a signingTime attribute is included with the response.
 - 2.d.2.b - Else if the glMemberName is not present on the GL, the GLA checks how the GL is administered.
 - 2.d.2.b.1 - If the GL is closed, the GLA checks that a registered GLO signed the request by checking that one of the names in the digital signature certificate used to sign the request matches a registered GLO.
 - 2.d.2.b.1.a - If the names do not match, the GLA returns a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of noGLONameMatch. Additionally, a signingTime attribute is included with the response.
 - 2.d.2.b.1.b - Else if the names match, the GLA verifies the member's encryption certificate.
 - 2.d.2.b.1.b.1 - If the member's encryption certificate cannot be verified, the GLA can return a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of invalidCert to the GLO.

Additionally, a `signingTime` attribute is included with the response. If the GLA does not return a `CMCStatusInfoExt.cMCStatus.failed` response, the GLA issues a `glProvideCert` request (see Section 4.10).

2.d.2.b.1.b.2 - Else if the member's certificate verifies, the GLA returns a `CMCStatusInfoExt` indicating `CMCStatus.success` and a `signingTime` attribute (2 in Figure 5). The GLA also takes administrative actions, which are beyond the scope of this document, to add the member to the GL stored on the GLA. The GLA also distributes the shared KEK to the member via the mechanism described in Section 5.

2.d.2.b.1.b.2.a - The GLA applies confidentiality to the response by encapsulating the `SignedData.PKIData` in an `EnvelopedData` if the request was encapsulated in an `EnvelopedData` (see Section 3.2.1.2).

2.d.2.b.1.b.2.b - The GLA can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).

2.d.2.b.2 - Else if the GL is managed, the GLA checks that either a registered GLO or the prospective member signed the request. For GLOs, one of the names in the certificate used to sign the request needs to match a registered GLO. For the prospective member, the name in `glMember.glMemberName` needs to match one of the names in the certificate used to sign the request.

2.d.2.b.2.a - If the signer is neither a registered GLO nor the prospective GL member, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `noSpam`. Additionally, a `signingTime` attribute is included with the response.

- 2.d.2.b.2.b - Else if the signer is a registered GLO, the GLA verifies the member's encryption certificate.
 - 2.d.2.b.2.b.1 - If the member's certificate cannot be verified, the GLA can return a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `invalidCert`. Additionally, a `signingTime` attribute is included with the response. If the GLA does not return a `CMCStatus.failed` response, the GLA MUST issue a `glProvideCert` request (see Section 4.10).
 - 2.d.2.b.2.b.2 - Else if the member's certificate verifies, the GLA MUST return a `CMCStatusInfoExt` indicating `CMCStatus.success` and a `signingTime` attribute to the GLO (2 in Figure 5). The GLA also takes administrative actions, which are beyond the scope of this document, to add the member to the GL stored on the GLA. The GLA also distributes the shared KEK to the member via the mechanism described in Section 5. The GL policy may mandate that the GL member's address be included in the GL member's certificate.
 - 2.d.2.b.2.b.2.a - The GLA applies confidentiality to the response by encapsulating the `SignedData.PKIData` in an `EnvelopedData` if the request was encapsulated in an `EnvelopedData` (see Section 3.2.1.2).
 - 2.d.2.b.2.b.2.b - The GLA can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
 - 2.d.2.b.2.c - Else if the signer is the prospective member, the GLA forwards the `glAddMember` request (see Section 3.2.3) to a registered GLO (B{A} in Figure 5). If there is more than one registered GLO, the GLO to which the request is forwarded is beyond the scope of this

document. Further processing of the forwarded request by GLOs is addressed in 3 of Section 4.3.2.

- 2.d.2.b.2.c.1 - The GLA applies confidentiality to the forwarded request by encapsulating the SignedData.PKIData in an EnvelopedData if the original request was encapsulated in an EnvelopedData (see Section 3.2.1.2).
- 2.d.2.b.2.c.2 - The GLA can also optionally apply another SignedData over the EnvelopedData (see Section 3.2.1.2).
- 2.d.2.b.3 - Else if the GL is unmanaged, the GLA checks that either a registered GLO or the prospective member signed the request. For GLOs, one of the names in the certificate used to sign the request needs to match the name of a registered GLO. For the prospective member, the name in glMember.glMemberName needs to match one of the names in the certificate used to sign the request.
 - 2.d.2.b.3.a - If the signer is neither a registered GLO nor the prospective member, the GLA returns a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of noSpam. Additionally, a signingTime attribute is included with the response.
 - 2.d.2.b.3.b - Else if the signer is either a registered GLO or the prospective member, the GLA verifies the member's encryption certificate.
 - 2.d.2.b.3.b.1 - If the member's certificate cannot be verified, the GLA can return a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of invalidCert and a signingTime attribute to either the GLO or the prospective member depending on where the request originated. If the GLA does not return a cMCStatus.failed response, the GLA issues a glProvideCert request (see

Section 4.10) to either the GLO or prospective member depending on where the request originated.

2.d.2.b.3.b.2 - Else if the member's certificate verifies, the GLA returns a `CMCStatusInfoExt` indicating `CMCStatus.success` and a `signingTime` attribute to the GLO (2 in Figure 5) if the GLO signed the request and to the GL member (3 in Figure 5) if the GL member signed the request. The GLA also takes administrative actions, which are beyond the scope of this document, to add the member to the GL stored on the GLA. The GLA also distributes the shared KEK to the member via the mechanism described in Section 5.

2.d.2.b.3.b.2.a - The GLA applies confidentiality to the response by encapsulating the `SignedData.PKIData` in an `EnvelopedData` if the request was encapsulated in an `EnvelopedData` (see Section 3.2.1.2).

2.d.2.b.3.b.2.b - The GLA can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).

3 - Upon receipt of the `CMCStatusInfoExt` response, the GLO checks the `signingTime` and verifies the GLA signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the response (see Section 3.2.1.2 or 3.2.2), the GLO verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.

3.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLO MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.

3.b - Else if signature processing continues and if the signatures verify, the GLO checks that one of the names in the certificate used to sign the response matches the name of the GL.

- 3.b.1 - If the name of the GL does not match the name present in the certificate used to sign the message, the GLO should not believe the response.
- 3.b.2 - Else if the name of the GL matches the name present in the certificate and:
 - 3.b.2.a - If the signatures verify and the response is `CMCStatusInfoExt` indicating `CMCStatus.success`, the GLA has added the member to the GL. If the member was added to a managed list and the original request was signed by the member, the GLO sends a `CMCStatusInfoExt.CMCStatus.success` and a `signingTime` attribute to the GL member.
 - 3.b.2.b - Else if the GLO received a `CMCStatusInfoExt.CMCStatus.failed` with any reason, the GLO can reattempt to add the member to the GL using the information provided in the response.
- 4 - Upon receipt of the `CMCStatusInfoExt` response, the prospective member checks the `signingTime` and verifies the GLA signatures or GLO signatures. If an additional `SignedData` and/or `EnvelopedData` encapsulates the response (see Section 3.2.1.2 or 3.2.2), the GLO verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 4.a - If the `signingTime` attribute value is not within the locally accepted time window, the prospective member MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
 - 4.b - Else if signature processing continues and if the signatures verify, the GL member checks that one of the names in the certificate used to sign the response matches the name of the GL.
 - 4.b.1 - If the name of the GL does not match the name present in the certificate used to sign the message, the GL member should not believe the response.
 - 4.b.2 - Else if the name of the GL matches the name present in the certificate and:
 - 4.b.2.a - If the signatures verify, the prospective member has been added to the GL.

- 4.b.2.b - Else if the prospective member received a `CMCStatusInfoExt.cMCStatus.failed`, for any reason, the prospective member MAY reattempt to add itself to the GL using the information provided in the response.

4.3.2. Prospective Member Initiated Additions

The process for prospective member initiated `glAddMember` requests is as follows:

- 1 - The prospective GL member sends a `SignedData.PKIData.controlSequence.glAddMember` request to the GLA (A in Figure 5). The prospective GL member includes: the GL name in `glName`, their name in `glMember.glMemberName`, their address in `glMember.glMemberAddress`, and their encryption certificate in `glMember.certificates.pKC`. The prospective GL member can also include any attribute certificates associated with their encryption certificate in `glMember.certificates.aC`, and the certification path associated with their encryption and attribute certificates in `glMember.certificates.certPath`. The prospective member MUST also include the `signingTime` attribute with this request.
 - 1.a - The prospective GL member can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` (see Section 3.2.1.2).
 - 1.b - The prospective GL member MAY optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 2 - Upon receipt of the request, the GLA verifies the request as per 2 in Section 4.3.1.
- 3 - Upon receipt of the forwarded request, the GLO checks the `signingTime` and verifies the prospective GL member signature on the innermost `SignedData.PKIData` and the GLA signature on the outer layer. If an `EnvelopedData` encapsulates the innermost layer (see Section 3.2.1.2 or 3.2.2), the GLO decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.

Note: For cases where the GL is closed and either a) a prospective member sends directly to the GLO or b) the GLA has mistakenly forwarded the request to the GLO, the GLO should first determine whether to honor the request.

- 3.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLO MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime`.
- 3.b - Else if signature processing continues and if the signatures verify, the GLO checks to make sure one of the names in the certificate used to sign the request matches the name in `glMember.glMemberName`.
 - 3.b.1 - If the names do not match, the GLO sends a `SignedData.PKIResponse.controlSequence` message back to the prospective member with `CMCStatusInfoExt.CMCStatus.failed` indicating why the prospective member was denied in `CMCStatusInfo.statusString`. This stops people from adding people to GLs without their permission. Additionally, a `signingTime` attribute is included with the response.
 - 3.b.2 - Else if the names match, the GLO determines whether the prospective member is allowed to be added. The mechanism is beyond the scope of this document; however, the GLO should check to see that the `glMember.glMemberName` is not already on the GL.
 - 3.b.2.a - If the GLO determines the prospective member is not allowed to join the GL, the GLO can return a `SignedData.PKIResponse.controlSequence` message back to the prospective member with `CMCStatusInfoExt.CMCStatus.failed` indicating why the prospective member was denied in `CMCStatus.statusString`. Additionally, a `signingTime` attribute is included with the response.
 - 3.b.2.b - Else if the GLO determines the prospective member is allowed to join the GL, the GLO verifies the member's encryption certificate.
 - 3.b.2.b.1 - If the member's certificate cannot be verified, the GLO returns a `SignedData.PKIResponse.controlSequence` back to the prospective member with `CMCStatusInfoExt.CMCStatus.failed` indicating that the member's encryption certificate did not verify in `CMCStatus.statusString`. Additionally, a `signingTime` attribute is included with the response. If the GLO does not return a `CMCStatusInfoExt` response, the GLO sends a

`SignedData.PKIData.controlSequence.glProvideCert` message to the prospective member requesting a new encryption certificate (see Section 4.10).

3.b.2.b.2 - Else if the member's certificate verifies, the GLO resubmits the `glAddMember` request (see Section 3.2.5) to the GLA (1 in Figure 5).

3.b.2.b.2.a - The GLO applies confidentiality to the new `glAddMember` request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` if the initial request was encapsulated in an `EnvelopedData` (see Section 3.2.1.2).

3.b.2.b.2.b - The GLO can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).

4 - Processing continues as in 2 of Section 4.3.1.

4.4. Delete Members from GL

To delete members from GLs, either the GLO or members to be removed use the `glDeleteMember` request. The GLA processes the GLO, and members requesting their own removal make requests differently. The GLO can submit the request at any time to delete members from the GL, and the GLA, once it has verified the request came from a registered GLO, should delete the member. If a member sends the request, the GLA needs to determine how the GL is administered. When the GLO initially configured the GL, it set the GL to be unmanaged, managed, or closed (see Section 3.1.1). In the unmanaged case, the GLA merely processes the member's request. In the managed case, the GLA forwards the requests from the member to the GLO for review. Where there are multiple GLOs for a GL, which GLO the request is forwarded to is beyond the scope of this document. The GLO reviews the request and either rejects it or submits a reformed request to the GLA. In the closed case, the GLA will not accept requests from members. The following sections describe the processing for the GLO(s), GLA, and GL members depending on where the request originated, either from a GLO or from members wanting to be removed. Figure 6 depicts the protocol interactions for the three options. Note that the error messages are not depicted. Additionally, behavior for the optional `transactionId`, `senderNonce`, and `recipientNonce` CMC control attributes is not addressed in these procedures.

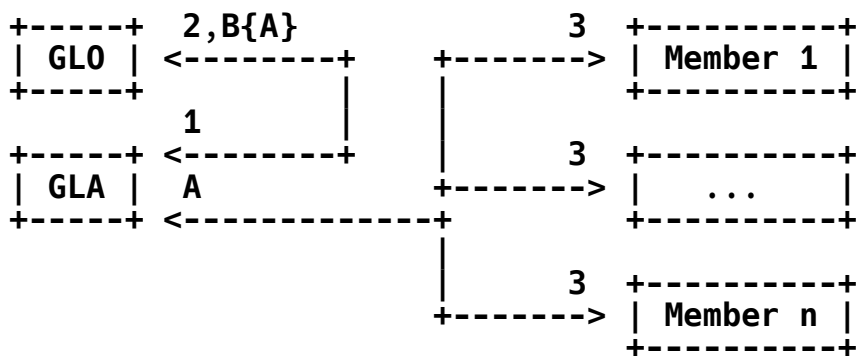


Figure 6 - Member Deletion

If the member is not removed from the GL, it will continue to receive and be able to decrypt data protected with the shared KEK and will continue to receive rekeys. For unmanaged lists, there is no point to a group rekey because there is no guarantee that the member requesting to be removed has not already added itself back on the GL under a different name. For managed and closed GLs, the GLO needs to take steps to ensure that the member being deleted is not on the GL twice. After ensuring this, managed and closed GLs can be rekeyed to maintain the confidentiality of the traffic sent by group members. If the GLO is sure the member has been deleted, the group rekey mechanism can be used to distribute the new key (see Sections 4.5 and 5).

4.4.1. GLO Initiated Deletions

The process for GLO initiated `glDeleteMember` requests is as follows:

- 1 - The GLO collects the pertinent information for the member(s) to be deleted (this can be done through an out-of-band means). The GLO then sends a `SignedData.PKIData.controlSequence` with a separate `glDeleteMember` request for each member to the GLA (1 in Figure 6). The GLO MUST include the GL name in `glName` and the member's name in `glMemberToDelete`. If the GL from which the member is being deleted is a closed or managed GL, the GLO MUST also generate a `glRekey` request and include it with the `glDeleteMember` request (see Section 4.5). The GLO MUST also include the `signingTime` attribute with this request.
 - 1.a - The GLO can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` (see Section 3.2.1.2).
 - 1.b - The GLO can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).

- 2 - Upon receipt of the request, the GLA checks the signingTime attribute and verifies the signature on the innermost SignedData.PKIData. If an additional SignedData and/or EnvelopedData encapsulates the request (see Section 3.2.1.2 or 3.2.2), the GLA verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost SignedData.
 - 2.a - If the signingTime attribute value is not within the locally accepted time window, the GLA MAY return a response indicating cMCStatus.failed and otherInfo.failInfo.badTime and a signingTime attribute.
 - 2.b - Else if signature processing continues and if the signatures cannot be verified, the GLA returns a cMCStatusInfoExt response indicating cMCStatus.failed and otherInfo.failInfo.badMessageCheck. Additionally, a signingTime attribute is included with the response.
 - 2.c - Else if the signatures verify, the GLA makes sure the GL is supported by the GLA by checking that the glName matches a glName stored on the GLA.
 - 2.c.1 - If the glName is not supported by the GLA, the GLA returns a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of invalidGLName. Additionally, a signingTime attribute is included with the response.
 - 2.c.2 - Else if the glName is supported by the GLA, the GLA checks to see if the glMemberName is present on the GL.
 - 2.c.2.a - If the glMemberName is not present on the GL, the GLA returns a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of notAMember. Additionally, a signingTime attribute is included with the response.
 - 2.c.2.b - Else if the glMemberName is already on the GL, the GLA checks how the GL is administered.
 - 2.c.2.b.1 - If the GL is closed, the GLA checks that the registered GLO signed the request by checking that one of the names in the digital signature certificate used to sign the request matches the registered GLO.

- 2.c.2.b.1.a - If the names do not match, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `closedGL`. Additionally, a `signingTime` attribute is included with the response.
- 2.c.2.b.1.b - Else if the names do match, the GLA returns a `CMCStatusInfoExt.CMCStatus.success` and a `signingTime` attribute (2 in Figure 5). The GLA also takes administrative actions, which are beyond the scope of this document, to delete the member with the GL stored on the GLA. Note that the GL also needs to be rekeyed as described in Section 5.
 - 2.c.2.b.1.b.1 - The GLA applies confidentiality to the response by encapsulating the `SignedData.PKIData` in an `EnvelopedData` if the request was encapsulated in an `EnvelopedData` (see Section 3.2.1.2).
 - 2.c.2.b.1.b.2 - The GLA can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 2.c.2.b.2 - Else if the GL is managed, the GLA checks that either a registered GLO or the prospective member signed the request. For GLOs, one of the names in the certificate used to sign the request needs to match a registered GLO. For the prospective member, the name in `glMember.glMemberName` needs to match one of the names in the certificate used to sign the request.
 - 2.c.2.b.2.a - If the signer is neither a registered GLO nor the prospective GL member, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `noSpam`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.2.b.2.b - Else if the signer is a registered GLO, the GLA returns a `CMCStatusInfoExt.CMCStatus.success` and a `signingTime` attribute (2 in Figure 6). The GLA also takes administrative actions, which

are beyond the scope of this document, to delete the member with the GL stored on the GLA. Note that the GL will also be rekeyed as described in Section 5.

- 2.c.2.b.2.b.1 - The GLA applies confidentiality to the response by encapsulating the SignedData.PKIData in an EnvelopedData if the request was encapsulated in an EnvelopedData (see Section 3.2.1.2).
- 2.c.2.b.2.b.2 - The GLA can also optionally apply another SignedData over the EnvelopedData (see Section 3.2.1.2).
- 2.c.2.b.2.c - Else if the signer is the prospective member, the GLA forwards the glDeleteMember request (see Section 3.2.3) to the GLO (B{A} in Figure 6). If there is more than one registered GLO, the GLO to which the request is forwarded to is beyond the scope of this document. Further processing of the forwarded request by GLOs is addressed in 3 of Section 4.4.2.
 - 2.c.2.b.2.c.1 - The GLA applies confidentiality to the forwarded request by encapsulating the SignedData.PKIData in an EnvelopedData if the request was encapsulated in an EnvelopedData (see Section 3.2.1.2).
 - 2.c.2.b.2.c.2 - The GLA can also optionally apply another SignedData over the EnvelopedData (see Section 3.2.1.2).
- 2.c.2.b.3 - Else if the GL is unmanaged, the GLA checks that either a registered GLO or the prospective member signed the request. For GLOs, one of the names in the certificate used to sign the request needs to match the name of a registered GLO. For the prospective member, the name in glMember.glMemberName needs to match one of the names in the certificate used to sign the request.

- 2.c.2.b.3.a - If the signer is neither the GLO nor the prospective member, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `noSpam`. Additionally, a `signingTime` attribute is included with the response.
- 2.c.2.b.3.b - Else if the signer is either a registered GLO or the member, the GLA returns a `CMCStatusInfoExt.CMCStatus.success` and a `signingTime` attribute to the GLO (2 in Figure 6) if the GLO signed the request and to the GL member (3 in Figure 6) if the GL member signed the request. The GLA also takes administrative actions, which are beyond the scope of this document, to delete the member with the GL stored on the GLA.
 - 2.c.2.b.3.b.1 - The GLA applies confidentiality to the response by encapsulating the `SignedData.PKIData` in an `EnvelopedData` if the request was encapsulated in an `EnvelopedData` (see Section 3.2.1.2).
 - 2.c.2.b.3.b.2 - The GLA can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 3 - Upon receipt of the `CMCStatusInfoExt` response, the GLO checks the `signingTime` and verifies the GLA signatures. If an additional `SignedData` and/or `EnvelopedData` encapsulates the response (see Section 3.2.1.2 or 3.2.2), the GLO verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 3.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLO MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
 - 3.b - Else if signature processing continues and if the signatures do verify, the GLO checks that one of the names in the certificate used to sign the response matches the name of the GL.

- 3.b.1 - If the name of the GL does not match the name present in the certificate used to sign the message, the GLO should not believe the response.
- 3.b.2 - Else if the name of the GL matches the name present in the certificate and:
 - 3.b.2.a - If the signatures verify and the response is `CMCStatusInfoExt.cMCStatus.success`, the GLO has deleted the member from the GL. If member was deleted from a managed list and the original request was signed by the member, the GLO sends a `CMCStatusInfoExt.cMCStatus.success` and a `signingTime` attribute to the GL member.
 - 3.b.2.b - Else if the GLO received a `CMCStatusInfoExt.cMCStatus.failed` with any reason, the GLO may reattempt to delete the member from the GL using the information provided in the response.
- 4 - Upon receipt of the `CMCStatusInfoExt` response, the member checks the `signingTime` and verifies the GLA signature(s) or GLO signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the response (see Section 3.2.1.2 or 3.2.2), the GLO verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 4.a - If the `signingTime` attribute value is not within the locally accepted time window, the prospective member MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
 - 4.b - Else if signature processing continues and if the signatures verify, the GL member checks that one of the names in the certificate used to sign the response matches the name of the GL.
 - 4.b.1 - If the name of the GL does not match the name present in the certificate used to sign the message, the GL member should not believe the response.
 - 4.b.2 - Else if the name of the GL matches the name present in the certificate and:
 - 4.b.2.a - If the signature(s) verify, the member has been deleted from the GL.

- 4.b.2.b - Else if the member received a `CMCStatusInfoExt.cMCStatus.failed` with any reason, the member can reattempt to delete itself from the GL using the information provided in the response.

4.4.2. Member Initiated Deletions

The process for member initiated deletion of its own membership using the `glDeleteMember` requests is as follows:

- 1 - The member sends a `SignedData.PKIData.controlSequence.glDeleteMember` request to the GLA (A in Figure 6). The member includes the name of the GL in `glName` and the member's own name in `glMemberToDelete`. The GL member MUST also include the `signingTime` attribute with this request.
 - 1.a - The member can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` (see Section 3.2.1.2).
 - 1.b - The member can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 2 - Upon receipt of the request, the GLA verifies the request as per 2 in Section 4.4.1.
- 3 - Upon receipt of the forwarded request, the GLO checks the `signingTime` and verifies the member signature on the innermost `SignedData.PKIData` and the GLA signature on the outer layer. If an `EnvelopedData` encapsulates the innermost layer (see Section 3.2.1.2 or 3.2.2), the GLO decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.

Note: For cases where the GL is closed and either (a) a prospective member sends directly to the GLO or (b) the GLA has mistakenly forwarded the request to the GLO, the GLO should first determine whether to honor the request.

- 3.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLO MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.

- 3.b - Else if signature processing continues if the signatures cannot be verified, the GLO returns a `cMCStatusInfoExt` response indicating `cMCStatus.failed` and `otherInfo.failInfo.badMessageCheck` and a `signingTime` attribute.
- 3.c - Else if the signatures verify, the GLO checks to make sure one of the names in the certificates used to sign the request matches the name in `glMemberToDelete`.
 - 3.c.1 - If the names do not match, the GLO sends a `SignedData.PKIResponse.controlSequence` message back to the prospective member with `cMCStatusInfoExt.cMCstatus.failed` indicating why the prospective member was denied in `cMCStatusInfoExt.statusString`. This stops people from adding people to GLs without their permission. Additionally, a `signingTime` attribute is included with the response.
 - 3.c.2 - Else if the names match, the GLO resubmits the `glDeleteMember` request (see Section 3.2.5) to the GLA (1 in Figure 6). The GLO makes sure the `glMemberName` is already on the GL. The GLO also generates a `glRekey` request and include it with the `glDeleteMember` request (see Section 4.5).
 - 3.c.2.a - The GLO applies confidentiality to the new `glDeleteMember` request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` if the initial request was encapsulated in an `EnvelopedData` (see Section 3.2.1.2).
 - 3.c.2.b - The GLO can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).

4 - Further processing is as in 2 of Section 4.4.1.

4.5. Request Rekey of GL

From time to time, the GL will need to be rekeyed. Some situations follow:

- When a member is removed from a closed or managed GL. In this case, the `PKIData.controlSequence` containing the `glDeleteMember` ought to contain a `glRekey` request.

- Depending on policy, when a member is removed from an unmanaged GL. If the policy is to rekey the GL, the PKIData.controlSequence containing the glDeleteMember could also contain a glRekey request or an out-of-bands means could be used to tell the GLA to rekey the GL. Rekeying of unmanaged GLs when members are deleted is not advised.
- When the current shared KEK has been compromised.
- When the current shared KEK is about to expire. Consider two cases:
 - If the GLO controls the GL rekey, the GLA should not assume that a new shared KEK should be distributed, but instead wait for the glRekey message.
 - If the GLA controls the GL rekey, the GLA should initiate a glKey message as specified in Section 5.

If the generationCounter (see Section 3.1.1) is set to a value greater than one (1) and the GLO controls the GL rekey, the GLO may generate a glRekey any time before the last shared KEK has expired. To be on the safe side, the GLO ought to request a rekey one (1) duration before the last shared KEK expires.

The GLA and GLO are the only entities allowed to initiate a GL rekey. The GLO indicated whether they are going to control rekeys or whether the GLA is going to control rekeys when they assigned the shared KEK to GL (see Section 3.1.1). The GLO initiates a GL rekey at any time. The GLA can be configured to automatically rekey the GL prior to the expiration of the shared KEK (the length of time before the expiration is an implementation decision). The GLA can also automatically rekey GLs that have been compromised, but this is covered in Section 5. Figure 7 depicts the protocol interactions to request a GL rekey. Note that error messages are not depicted. Additionally, behavior for the optional transactionId, senderNonce, and recipientNonce CMC control attributes is not addressed in these procedures.

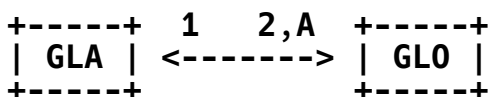


Figure 7 - GL Rekey Request

4.5.1. GLO Initiated Rekey Requests

The process for GLO initiated glRekey requests is as follows:

- 1 - The GLO sends a SignedData.PKIData.controlSequence.glRekey request to the GLA (1 in Figure 7). The GLO includes the glName. If glAdministration and glKeyNewAttributes are omitted then there is no change from the previously registered GL values for these fields. If the GLO wants to force a rekey for all outstanding shared KEKs, it includes the glRekeyAllGLKeys set to TRUE. The GLO MUST also include a signingTime attribute with this request.
 - 1.a - The GLO can optionally apply confidentiality to the request by encapsulating the SignedData.PKIData in an EnvelopedData (see Section 3.2.1.2).
 - 1.b - The GLO can also optionally apply another SignedData over the EnvelopedData (see Section 3.2.1.2).
- 2 - Upon receipt of the request, the GLA checks the signingTime and verifies the signature on the innermost SignedData.PKIData. If an additional SignedData and/or EnvelopedData encapsulates the request (see Section 3.2.1.2 or 3.2.2), the GLA verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost SignedData.
 - 2.a - If the signingTime attribute value is not within the locally accepted time window, the GLA MAY return a response indicating cMCStatus.failed and otherInfo.failInfo.badTime and a signingTime attribute.
 - 2.b - Else if signature processing continues and if the signatures do not verify, the GLA returns a cMCStatusInfoExt response indicating cMCStatus.failed and otherInfo.failInfo.badMessageCheck. Additionally, a signingTime attribute is included with the response.
 - 2.c - Else if the signatures do verify, the GLA makes sure the GL is supported by the GLA by checking that the glName matches a glName stored on the GLA.
 - 2.c.1 - If the glName present does not match a GL stored on the GLA, the GLA returns a response indicating cMCStatusInfoExt with cMCStatus.failed and otherInfo.extendedFailInfo.SKDFailInfo value of invalidGLName. Additionally, a signingTime attribute is included with the response.

- 2.c.2 - Else if the `glName` present matches a GL stored on the GLA, the GLA checks that a registered GLO signed the request by checking that one of the names in the certificate used to sign the request is a registered GLO.
 - 2.c.2.a - If the names do not match, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `noGLONameMatch`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.2.b - Else if the names match, the GLA checks the `glNewKeyAttribute` values.
 - 2.c.2.b.1 - If the new value for `requestedAlgorithm` is not supported, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `unsupportedAlgorithm`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.2.b.2 - Else if the new value duration is not supportable (determining this is beyond the scope of this document), the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `unsupportedDuration`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.2.b.3 - Else if the GL is not supportable for other reasons that the GLA does not wish to disclose, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `unspecified`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.2.b.4 - Else if the new `requestedAlgorithm` and duration are supportable or the `glNewKeyAttributes` was omitted, the GLA returns a `CMCStatusInfoExt.CMCStatus.success` and a `signingTime` attribute (2 in Figure 7). The GLA also uses the `glKey` message to distribute the rekey shared KEK (see Section 5).

- 2.c.2.b.4.a - The GLA applies confidentiality to response by encapsulating the SignedData.PKIData in an EnvelopedData if the request was encapsulated in an EnvelopedData (see Section 3.2.1.2).
 - 2.c.2.b.4.b - The GLA can also optionally apply another SignedData over the EnvelopedData (see Section 3.2.1.2).
- 3 - Upon receipt of the cMCStatusInfoExt response, the GLO checks the signingTime and verifies the GLA signature(s). If an additional SignedData and/or EnvelopedData encapsulates the forwarded response (see Section 3.2.1.2 or 3.2.2), the GLO verifies the outer signature and/or decrypts the forwarded response prior to verifying the signature on the innermost SignedData.
- 3.a - If the signingTime attribute value is not within the locally accepted time window, the GLA MAY return a response indicating cMCStatus.failed and otherInfo.failInfo.badTime and a signingTime attribute.
 - 3.b - Else if signature processing continues and if the signatures verify, the GLO checks that one of the names in the certificate used to sign the response matches the name of the GL.
 - 3.b.1 - If the name of the GL does not match the name present in the certificate used to sign the message, the GLO should not believe the response.
 - 3.b.2 - Else if the name of the GL matches the name present in the certificate and:
 - 3.b.2.a - If the signatures verify and the response is cMCStatusInfoExt.cMCStatus.success, the GLO has successfully rekeyed the GL.
 - 3.b.2.b - Else if the GLO received a cMCStatusInfoExt.cMCStatus.failed with any reason, the GLO can reattempt to rekey the GL using the information provided in the response.

4.5.2. GLA Initiated Rekey Requests

If the GLA is in charge of rekeying the GL the GLA will automatically issue a `glKey` message (see Section 5). In addition the GLA will generate a `cMCStatusInfoExt` to indicate to the GL that a successful rekey has occurred. The process for GLA initiated rekey is as follows:

- 1 - The GLA generates for all GLOs a `SignedData.PKIData.controlSequence.cMCStatusInfoExt.cMCStatus` success and includes a `signingTime` attribute (A in Figure 7).
 - 1.a - The GLA can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` (see Section 3.2.1.2).
 - 1.b - The GLA can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 2 - Upon receipt of the `cMCStatusInfoExt.cMCStatus.success` response, the GLO checks the `signingTime` and verifies the GLA signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the forwarded response (see Section 3.2.1.2 or 3.2.2), the GLO MUST verify the outer signature and/or decrypt the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 2.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLO MAY return a response indicating `cMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
 - 2.b - Else if signature processing continues and if the signatures verify, the GLO checks that one of the names in the certificate used to sign the response matches the name of the GL.
 - 2.b.1 - If the name of the GL does not match the name present in the certificate used to sign the message, the GLO ought not believe the response.
 - 2.b.2 - Else if the name of the GL does match the name present in the certificate and the response is `cMCStatusInfoExt.cMCStatus.success`, the GLO knows the GLA has successfully rekeyed the GL.

4.6. Change GLO

Management of managed and closed GLs can become difficult for one GLO if the GL membership grows large. To support distributing the workload, GLAs support having GLs be managed by multiple GLOs. The `glAddOwner` and `glRemoveOwner` messages are designed to support adding and removing registered GLOs. Figure 8 depicts the protocol interactions to send `glAddOwner` and `glRemoveOwner` messages and the resulting response messages. Note that error messages are not shown. Additionally, behavior for the optional `transactionId`, `senderNonce`, and `recipientNonce` CMC control attributes is not addressed in these procedures.

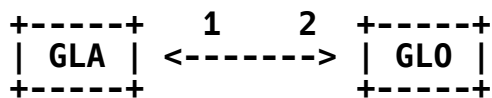


Figure 8 - GLO Add and Delete Owners

The process for `glAddOwner` and `glDeleteOwner` is as follows:

- 1 - The GLO sends a `SignedData.PKIData.controlSequence.glAddOwner` or `glRemoveOwner` request to the GLA (1 in Figure 8). The GLO includes the GL name in `glName`, and the name and address of the GLO in `glOwnerName` and `glOwnerAddress`, respectively. The GLO MUST also include the `signingTime` attribute with this request.
 - 1.a - The GLO can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` (see Section 3.2.1.2).
 - 1.b - The GLO can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 2 - Upon receipt of the `glAddOwner` or `glRemoveOwner` request, the GLA checks the `signingTime` and verifies the GLO signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the request (see Section 3.2.1.2 or 3.2.2), the GLA verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 2.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLA MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.

- 2.b - Else if signature processing continues and if the signatures cannot be verified, the GLA returns a `CMCStatusInfoExt` response indicating `CMCStatus.failed` and `otherInfo.failInfo.badMessageCheck`. Additionally, a `signingTime` attribute is included with the response.
- 2.c - Else if the signatures verify, the GLA makes sure the GL is supported by checking that the `glName` matches a `glName` stored on the GLA.
 - 2.c.1 - If the `glName` is not supported by the GLA, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `invalidGLName`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.2 - Else if the `glName` is supported by the GLA, the GLA ensures that a registered GLO signed the `glAddOwner` or `glRemoveOwner` request by checking that one of the names present in the digital signature certificate used to sign the `glAddOwner` or `glDeleteOwner` request matches the name of a registered GLO.
 - 2.c.2.a - If the names do not match, the GLA returns a response indicating `CMCStatusInfoExt` with `CMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `noGLONameMatch`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.2.b - Else if the names match, the GLA returns a `CMCStatusInfoExt.CMCStatus.success` and a `signingTime` attribute (2 in Figure 4). The GLA also takes administrative actions to associate the new `glOwnerName` with the GL in the case of `glAddOwner` or to disassociate the old `glOwnerName` with the GL in the case of `glRemoveOwner`.
 - 2.c.2.b.1 - The GLA applies confidentiality to the response by encapsulating the `SignedData.PKIResponse` in an `EnvelopedData` if the request was encapsulated in an `EnvelopedData` (see Section 3.2.1.2).
 - 2.c.2.b.2 - The GLA can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).

- 3 - Upon receipt of the `CMCStatusInfoExt` response, the GLO checks the `signingTime` and verifies the GLA's signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the response (see Section 3.2.1.2 or 3.2.2), the GLO verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 3.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLO MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
 - 3.b - Else if signature processing continues and if the signatures verify, the GLO checks that one of the names in the certificate used to sign the response matches the name of the GL.
 - 3.b.1 - If the name of the GL does not match the name present in the certificate used to sign the message, the GLO should not believe the response.
 - 3.b.2 - Else if the name of the GL does match the name present in the certificate and:
 - 3.b.2.a - If the signatures verify and the response was `CMCStatusInfoExt.CMCStatus.success`, the GLO has successfully added or removed the GLO.
 - 3.b.2.b - Else if the signatures verify and the response was `CMCStatusInfoExt.CMCStatus.failed` with any reason, the GLO can reattempt to add or delete the GLO using the information provided in the response.

4.7. Indicate KEK Compromise

There will be times when the shared KEK is compromised. GL members and GLOs use `glkCompromise` to tell the GLA that the shared KEK has been compromised. Figure 9 depicts the protocol interactions for GL Key Compromise. Note that error messages are not shown. Additionally, behavior for the optional `transactionId`, `senderNonce`, and `recipientNonce` CMC control attributes is not addressed in these procedures.

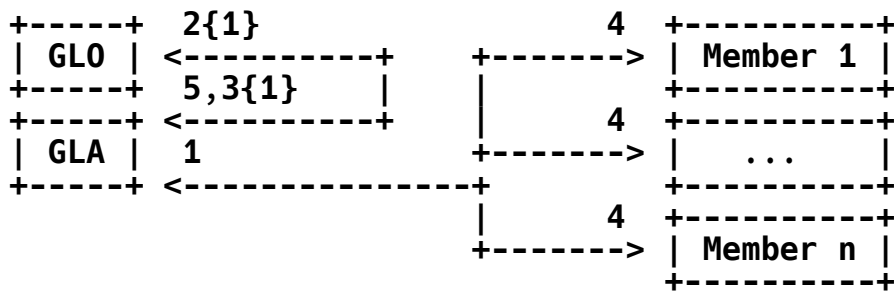


Figure 9 - GL Key Compromise

4.7.1. GL Member Initiated KEK Compromise Message

The process for GL member initiated glkCompromise messages is as follows:

- 1 - The GL member sends a SignedData.PKIData.controlSequence.glkCompromise request to the GLA (1 in Figure 9). The GL member includes the name of the GL in GeneralName. The GL member MUST also include the signingTime attribute with this request.
 - 1.a - The GL member can optionally apply confidentiality to the request by encapsulating the SignedData.PKIData in an EnvelopedData (see Section 3.2.1.2). The glkCompromise can be included in an EnvelopedData generated with the compromised shared KEK.
 - 1.b - The GL member can also optionally apply another SignedData over the EnvelopedData (see Section 3.2.1.2).
- 2 - Upon receipt of the glkCompromise request, the GLA checks the signingTime and verifies the GL member signature(s). If an additional SignedData and/or EnvelopedData encapsulates the request (see Section 3.2.1.2 or 3.2.2), the GLA verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost SignedData.
 - 2.a - If the signingTime attribute value is not within the locally accepted time window, the GLA MAY return a response indicating cMCStatus.failed and otherInfo.failInfo.badTime and a signingTime attribute.

- 2.b - Else if signature processing continues and if the signatures cannot be verified, the GLA returns a `cMCStatusInfoExt` response indicating `cMCStatus.failed` and `otherInfo.failInfo.badMessageCheck`. Additionally, a `signingTime` attribute is included with the response.
- 2.c - Else if the signatures verify, the GLA makes sure the GL is supported by checking that the indicated GL name matches a `glName` stored on the GLA.
 - 2.c.1 - If the `glName` is not supported by the GLA, the GLA returns a response indicating `cMCStatusInfoExt` with `cMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `invalidGLName`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.2 - Else if the `glName` is supported by the GLA, the GLA checks who signed the request. For GLOs, one of the names in the certificate used to sign the request needs to match a registered GLO. For the member, the name in `glMember.glMemberName` needs to match one of the names in the certificate used to sign the request.
 - 2.c.2.a - If the GLO signed the request, the GLA generates a `glKey` message as described in Section 5 to rekey the GL (4 in Figure 9).
 - 2.c.2.b - Else if someone other than the GLO signed the request, the GLA forwards the `glkCompromise` message (see Section 3.2.3) to the GLO (2{1} in Figure 9). If there is more than one GLO, to which GLO the request is forwarded is beyond the scope of this document. Further processing by the GLO is discussed in Section 4.7.2.

4.7.2. GLO Initiated KEK Compromise Message

The process for GLO initiated `glkCompromise` messages is as follows:

1 - The GLO either:

- 1.a - Generates the `glkCompromise` message itself by sending a `SignedData.PKIData.controlSequence.glkCompromise` request to the GLA (5 in Figure 9). The GLO includes the name of the GL in `GeneralName`. The GLO MUST also include a `signingTime` attribute with this request.

- 1.a.1 - The GLO can optionally apply confidentiality to the request by encapsulating the SignedData.PKIData in an EnvelopedData (see Section 3.2.1.2). The glkCompromise can be included in an EnvelopedData generated with the compromised shared KEK.
- 1.a.2 - The GLO can also optionally apply another SignedData over the EnvelopedData (see Section 3.2.1.2).
- 1.b - Otherwise, checks the signingTime and verifies the GLA and GL member signatures on the forwarded glkCompromise message. If an additional SignedData and/or EnvelopedData encapsulates the request (see Section 3.2.1.2 or 3.2.2), the GLO verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost SignedData.
 - 1.b.1 - If the signingTime attribute value is not within the locally accepted time window, the GLO MAY return a response indicating cMCStatus.failed and otherInfo.failInfo.badTime and a signingTime attribute.
 - 1.b.2 - Else if signature processing continues and if the signatures cannot be verified, the GLO returns a cMCStatusInfoExt response indicating cMCStatus.failed and otherInfo.failInfo.badMessageCheck. Additionally, a signingTime attribute is included with the response.
 - 1.b.2.a - If the signatures verify, the GLO checks that the names in the certificate match the name of the signer (i.e., the name in the certificate used to sign the GL member's request is the GL member).
 - 1.b.2.a.1 - If either name does not match, the GLO ought not trust the signer and it ought not forward the message to the GLA.
 - 1.b.2.a.2 - Else if the names match and the signatures verify, the GLO determines whether to forward the glkCompromise message back to the GLA (3{1} in Figure 9). Further processing by the GLA is in 2 of Section 4.7.1. The GLO can also return a response to the prospective member with cMCStatusInfoExt.cMCtatus.success indicating that the glkCompromise message was successfully received.

4.8. Request KEK Refresh

There will be times when GL members have irrecoverably lost their shared KEK. The shared KEK is not compromised and a rekey of the entire GL is not necessary. GL members use the `glkRefresh` message to request that the shared KEK(s) be redistributed to them. Figure 10 depicts the protocol interactions for GL Key Refresh. Note that error messages are not shown. Additionally, behavior for the optional `transactionId`, `senderNonce`, and `recipientNonce` CMC control attributes is not addressed in these procedures.

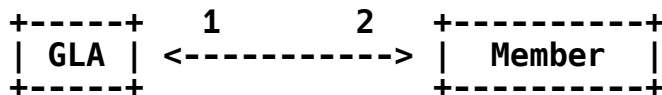


Figure 10 - GL KEK Refresh

The process for `glkRefresh` is as follows:

- 1 - The GL member sends a `SignedData.PKIData.controlSequence.glkRefresh` request to the GLA (1 in Figure 10). The GL member includes name of the GL in `GeneralName`. The GL member **MUST** also include a `signingTime` attribute with this request.
 - 1.a - The GL member can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` (see Section 3.2.1.2).
 - 1.b - The GL member can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 2 - Upon receipt of the `glkRefresh` request, the GLA checks the `signingTime` and verifies the GL member signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the request (see Section 3.2.1.2 or 3.2.2), the GLA verifies the outer signature and/or decrypt the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 2.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLA **MAY** return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.

- 2.b - Else if signature processing continues and if the signatures cannot be verified, the GLA returns a `cMCStatusInfoExt` response indicating `cMCStatus.failed` and `otherInfo.failInfo.badMessageCheck`. Additionally, a `signingTime` attribute is included with the response.
- 2.c - Else if the signatures verify, the GLA makes sure the GL is supported by checking that the `GLGeneralName` matches a `glName` stored on the GLA.
 - 2.c.1 - If the name of the GL is not supported by the GLA, the GLA returns a response indicating `cMCStatusInfoExt` with `cMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `invalidGLName`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.2 - Else if the `glName` is supported by the GLA, the GLA ensures that the GL member is on the GL.
 - 2.c.2.a - If the `glMemberName` is not present on the GL, the GLA returns a response indicating `cMCStatusInfoExt` with `cMCStatus.failed` and `otherInfo.extendedFailInfo.SKDFailInfo` value of `noSpam`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.2.b - Else if the `glMemberName` is present on the GL, the GLA returns a `cMCStatusInfoExt.cMCStatus.success`, a `signingTime` attribute, and a `glKey` message (2 in Figure 10) as described in Section 5.

4.9. GLA Query Request and Response

There will be certain times when a GLO is having trouble setting up a GL because it does not know the algorithm(s) or some other characteristic that the GLA supports. There can also be times when prospective GL members or GL members need to know something about the GLA (these requests are not defined in the document). The `glaQueryRequest` and `glaQueryResponse` messages have been defined to support determining this information. Figure 11 depicts the protocol interactions for `glaQueryRequest` and `glaQueryResponse`. Note that error messages are not shown. Additionally, behavior for the optional `transactionId`, `senderNonce`, and `recipientNonce` CMC control attributes is not addressed in these procedures.

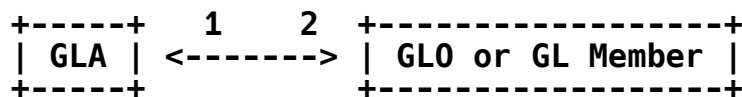


Figure 11 - GLA Query Request and Response

The process for `glaQueryRequest` and `glaQueryResponse` is as follows:

- 1 - The GLO, GL member, or prospective GL member sends a `SignedData.PKIData.controlSequence.glaQueryRequest` request to the GLA (1 in Figure 11). The GLO, GL member, or prospective GL member indicates the information it is interested in receiving from the GLA. Additionally, a `signingTime` attribute is included with this request.
 - 1.a - The GLO, GL member, or prospective GL member can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` (see Section 3.2.1.2).
 - 1.b - The GLO, GL member, or prospective GL member can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 2 - Upon receipt of the `glaQueryRequest`, the GLA determines if it accepts `glaQueryRequest` messages.
 - 2.a - If the GLA does not accept `glaQueryRequest` messages, the GLA returns a `cMCStatusInfoExt` response indicating `cMCStatus.noSupport` and any other information in `statusString`.
 - 2.b - Else if the GLA does accept `GLAQueryRequests`, the GLA checks the `signingTime` and verifies the GLO, GL member, or prospective GL member signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the request (see Section 3.2.1.2 or 3.2.2), the GLA verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 2.b.1 - If the `signingTime` attribute value is not within the locally accepted time window, the GLA MAY return a response indicating `cMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.

- 2.b.2 - Else if the signature processing continues and if the signatures cannot be verified, the GLA returns a `CMCStatusInfoExt` response indicating `CMCStatus.failed` and `otherInfo.failInfo.badMessageCheck`. Additionally, a `signingTime` attribute is included with the response.
- 2.b.3 - Else if the signatures verify, the GLA returns a `glaQueryResponse` (2 in Figure 11) with the correct response if the `glaRequestType` is supported or returns a `CMCStatusInfoExt` response indicating `CMCStatus.noSupport` if the `glaRequestType` is not supported. Additionally, a `signingTime` attribute is included with the response.
 - 2.b.3.a - The GLA applies confidentiality to the response by encapsulating the `SignedData.PKIResponse` in an `EnvelopedData` if the request was encapsulated in an `EnvelopedData` (see Section 3.2.1.2).
 - 2.b.3.b - The GLA can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 3 - Upon receipt of the `glaQueryResponse`, the GLO, GL member, or prospective GL member checks the `signingTime` and verifies the GLA signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the response (see Section 3.2.1.2 or 3.2.2), the GLO, GL member, or prospective GL member verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 3.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLO, GL member, or prospective GL member MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
 - 3.b - Else if signature processing continues and if the signatures do not verify, the GLO, GL member, or prospective GL member returns a `CMCStatusInfoExt` response indicating `CMCStatus.failed` and `otherInfo.failInfo.badMessageCheck`. Additionally, a `signingTime` attribute is included with the response.
 - 3.c - Else if the signatures verify, then the GLO, GL member, or prospective GL member checks that one of the names in the certificate used to sign the response matches the name of the GL.

- 3.c.1 - If the name of the GL does not match the name present in the certificate used to sign the message, the GLO ought not believe the response.
- 3.c.2 - Else if the name of the GL matches the name present in the certificate and the response was `glaQueryResponse`, then the GLO, GL member, or prospective GL member may use the information contained therein.

4.10. Update Member Certificate

When the GLO generates a `glAddMember` request, when the GLA generates a `glKey` message, or when the GLA processes a `glAddMember`, there can be instances when the GL member's certificate has expired or is invalid. In these instances, the GLO or GLA may request that the GL member provide a new certificate to avoid the GLA from being unable to generate a `glKey` message for the GL member. There might also be times when the GL member knows that its certificate is about to expire or has been revoked, and GL member will not be able to receive GL rekeys. Behavior for the optional `transactionId`, `senderNonce`, and `recipientNonce` CMC control attributes is not addressed in these procedures.

4.10.1. GLO and GLA Initiated Update Member Certificate

The process for GLO initiated `glUpdateCert` is as follows:

- 1 - The GLO or GLA sends a `SignedData.PKIData.controlSequence.glProvideCert` request to the GL member. The GLO or GLA indicates the GL name in `glName` and the GL member name in `glMemberName`. Additionally, a `signingTime` attribute is included with this request.
 - 1.a - The GLO or GLA can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` (see Section 3.2.1.2). If the GL member's PKC has been revoked, the GLO or GLA ought not use it to generate the `EnvelopedData` that encapsulates the `glProvideCert` request.
 - 1.b - The GLO or GLA can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).

- 2 - Upon receipt of the `glProvideCert` message, the GL member checks the `signingTime` and verifies the GLO or GLA signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the response (see Section 3.2.1.2 or 3.2.2), the GL member verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 2.a - If the `signingTime` attribute value is not within the locally accepted time window, the GL member MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
 - 2.b - Else if signature processing continues and if the signatures cannot be verified, the GL member returns a `CMCStatusInfoExt` response indicating `CMCStatus.failed` and `otherInfo.failInfo.badMessageCheck`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c - Else if the signatures verify, the GL member generates a `Signed.PKIResponse.controlSequence.glUpdateCert` that includes the GL name in `glName`, the member's name in `glMember.glMemberName`, the member's encryption certificate in `glMember.certificates.pKC`. The GL member can also include any attribute certificates associated with the member's encryption certificate in `glMember.certificates.aC`, and the certification path associated with the member's encryption and attribute certificates in `glMember.certificates.certPath`. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.1 - The GL member can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIResponse` in an `EnvelopedData` (see Section 3.2.1.2). If the GL member's PKC has been revoked, the GL member ought not use it to generate the `EnvelopedData` that encapsulates the `glProvideCert` request.
 - 2.c.2 - The GL member can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 3 - Upon receipt of the `glUpdateCert` message, the GLO or GLA checks the `signingTime` and verifies the GL member signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the response (see Section 3.2.1.2 or 3.2.2), the GL member verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.

- 3.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLO or GLA MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
- 3.b - Else if signature processing continues and if the signatures cannot be verified, the GLO or GLA returns a `CMCStatusInfoExt` response indicating `CMCStatus.failed` and `otherInfo.failInfo.badMessageCheck`. Additionally, a `signingTime` attribute is included with the response.
- 3.c - Else if the signatures verify, the GLO or GLA verifies the member's encryption certificate.
 - 3.c.1 - If the member's encryption certificate cannot be verified, the GLO returns either another `glProvideCert` request or a `CMCStatusInfoExt` with `CMCStatus.failed` and the reason why in `CMCStatus.statusString`. `glProvideCert` should be returned only a certain number of times is because if the GL member does not have a valid certificate it will never be able to return one. Additionally, a `signingTime` attribute is included with either response.
 - 3.c.2 - Else if the member's encryption certificate cannot be verified, the GLA returns another `glProvideCert` request to the GL member or a `CMCStatusInfoExt` with `CMCStatus.failed` and the reason why in `CMCStatus.statusString` to the GLO. `glProvideCert` should be returned only a certain number of times because if the GL member does not have a valid certificate it will never be able to return one. Additionally, a `signingTime` attribute is included with the response.
 - 3.c.3 - Else if the member's encryption certificate verifies, the GLO or GLA will use it in subsequent `glAddMember` requests and `glKey` messages associated with the GL member.

4.10.2. GL Member Initiated Update Member Certificate

The process for an unsolicited GL member `glUpdateCert` is as follows:

- 1 - The GL member sends a `Signed.PKIData.controlSequence.glUpdateCert` that includes the GL name in `glName`, the member's name in `glMember.glMemberName`, the member's encryption certificate in `glMember.certificates.pKC`. The GL member can also include any attribute certificates associated with the member's encryption certificate in `glMember.certificates.aC`, and the certification

path associated with the member's encryption and attribute certificates in `glMember.certificates.certPath`. The GL member MUST also include a `signingTime` attribute with this request.

- 1.a - The GL member can optionally apply confidentiality to the request by encapsulating the `SignedData.PKIData` in an `EnvelopedData` (see Section 3.2.1.2). If the GL member's PKC has been revoked, the GLO or GLA ought not use it to generate the `EnvelopedData` that encapsulates the `glProvideCert` request.
- 1.b - The GL member can also optionally apply another `SignedData` over the `EnvelopedData` (see Section 3.2.1.2).
- 2 - Upon receipt of the `glUpdateCert` message, the GLA checks the `signingTime` and verifies the GL member signature(s). If an additional `SignedData` and/or `EnvelopedData` encapsulates the response (see Section 3.2.1.2 or 3.2.2), the GLA verifies the outer signature and/or decrypts the outer layer prior to verifying the signature on the innermost `SignedData`.
 - 2.a - If the `signingTime` attribute value is not within the locally accepted time window, the GLA MAY return a response indicating `CMCStatus.failed` and `otherInfo.failInfo.badTime` and a `signingTime` attribute.
 - 2.b - Else if signature processing continues and if the signatures cannot be verified, the GLA returns a `CMCStatusInfoExt` response indicating `CMCStatus.failed` and `otherInfo.failInfo.badMessageCheck`.
 - 2.c - Else if the signatures verify, the GLA verifies the member's encryption certificate.
 - 2.c.1 - If the member's encryption certificate cannot be verified, the GLA returns another `glProvideCert` request to the GL member or a `CMCStatusInfoExt` with `CMCStatus.failed` and the reason why in `CMCStatus.statusString` to the GLO. `glProvideCert` ought not be returned indefinitely; if the GL member does not have a valid certificate it will never be able to return one. Additionally, a `signingTime` attribute is included with the response.
 - 2.c.2 - Else if the member's encryption certificate verifies, the GLA will use it in subsequent `glAddMember` requests and `glKey` messages associated with the GL member. The GLA also forwards the `glUpdateCert` message to the GLO.

5. Distribution Message

The GLA uses the `glKey` message to distribute new, shared KEK(s) after receiving `glAddMember`, `glDeleteMember` (for closed and managed GLs), `glRekey`, `glkCompromise`, or `glkRefresh` requests and returning a `cmCStatusInfoExt` response for the respective request. Figure 12 depicts the protocol interactions to send out `glKey` messages. Unlike the procedures defined for the administrative messages, the procedures defined in this section **MUST** be implemented by GLAs for origination and by GL members on reception. Note that error messages are not shown. Additionally, behavior for the optional `transactionId`, `senderNonce`, and `recipientNonce` CMC control attributes is not addressed in these procedures.

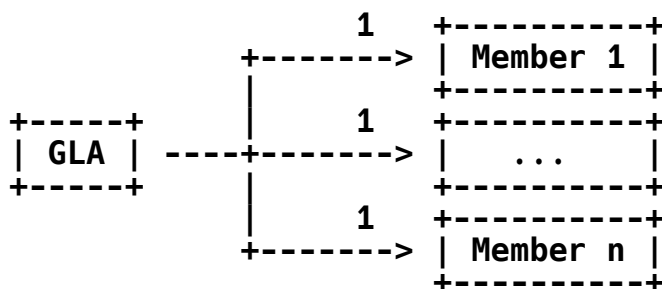


Figure 12 - GL Key Distribution

If the GL was set up with `GLKeyAttributes.recipientsNotMutuallyAware` set to `TRUE`, a separate `glKey` message **MUST** be sent to each GL member so as not to divulge information about the other GL members.

When the `glKey` message is generated as a result of a:

- `glAddMember` request,
- `glkCompromise` indication,
- `glkRefresh` request,
- `glDeleteMember` request with the GL's `glAdministration` set to managed or closed, and
- `glRekey` request with `generationCounter` set to zero (0).

The GLA **MUST** use either the `kari` (see Section 12.3.2 of [CMS]) or `ktri` (see Section 12.3.1 of [CMS]) choice in `glKey.glkWrapped.RecipientInfo` to ensure that only the intended recipients receive the shared KEK. The GLA **MUST** support the `ktri` choice.

When the glKey message is generated as a result of a glRekey request with generationCounter greater than zero (0) or when the GLA controls rekeys, the GLA MAY use the kari, ktri, or kekri (see Section 12.3.3 of [CMS]) in glKey.glkWrapped.RecipientInfo to ensure that only the intended recipients receive the shared KEK. The GLA MUST support the RecipientInfo.ktri choice.

5.1. Distribution Process

When a glKey message is generated, the process is as follows:

- 1 - The GLA MUST send a SignedData.PKIData.controlSequence.glKey to each member by including glName, glIdentifier, glkWrapped, glkAlgorithm, glkNotBefore, and glkNotAfter. If the GLA cannot generate a glKey message for the GL member because the GL member's PKC has expired or is otherwise invalid, the GLA MAY send a glUpdateCert to the GL member requesting a new certificate be provided (see Section 4.10). The number of glKey messages generated for the GL is described in Section 3.1.13. Additionally, a signingTime attribute is included with the distribution message(s).
 - 1.a - The GLA MAY optionally apply another confidentiality layer to the message by encapsulating the SignedData.PKIData in another EnvelopedData (see Section 3.2.1.2).
 - 1.b - The GLA MAY also optionally apply another SignedData over the EnvelopedData.SignedData.PKIData (see Section 3.2.1.2).
- 2 - Upon receipt of the glKey message, the GL members MUST check the signingTime and verify the signature over the innermost SignedData.PKIData. If an additional SignedData and/or EnvelopedData encapsulates the message (see Section 3.2.1.2 or 3.2.2), the GL member MUST verify the outer signature and/or decrypt the outer layer prior to verifying the signature on the SignedData.PKIData.controlSequence.glKey.
 - 2.a - If the signingTime attribute value is not within the locally accepted time window, the GLA MAY return a response indicating cMCStatus.failed and otherInfo.failInfo.badTime and a signingTime attribute.
 - 2.b - Else if signature processing continues and if the signatures cannot be verified, the GL member MUST return a cMCStatusInfoExt response indicating cMCStatus.failed and otherInfo.failInfo.badMessageCheck. Additionally, a signingTime attribute is included with the response.

- 2.c - Else if the signatures verify, the GL member processes the RecipientInfos according to [CMS]. Once unwrapped, the GL member should store the shared KEK in a safe place. When stored, the glName, glIdentifier, and shared KEK should be associated. Additionally, the GL member MUST return a cMCStatusInfoExt indicating cMCStatus.success to tell the GLA the KEK was received.

6. Algorithms

This section lists the algorithms that MUST be implemented. Additional algorithms that SHOULD be implemented are also included. Further algorithms MAY also be implemented.

6.1. KEK Generation Algorithm

Implementations MUST randomly generate content-encryption keys, message-authentication keys, initialization vectors (IVs), and padding. Also, the generation of public/private key pairs relies on a random numbers. The use of inadequate pseudo-random number generators (PRNGs) to generate cryptographic keys can result in little or no security. An attacker may find it much easier to reproduce the PRNG environment that produced the keys, searching the resulting small set of possibilities, rather than brute force searching the whole key space. The generation of quality random numbers is difficult. RFC 4086 [RANDOM] offers important guidance in this area, and Appendix 3 of FIPS Pub 186 [FIPS] provides one quality PRNG technique.

6.2. Shared KEK Wrap Algorithm

In the mechanisms described in Section 5, the shared KEK being distributed in glkWrapped MUST be protected by a key of equal or greater length (e.g., if an AES 128-bit key is being distributed, a key of 128 bits or greater must be used to protect the key).

The algorithm object identifiers included in glkWrapped are as specified in [CMSALG] and [CMSAES].

6.3. Shared KEK Algorithm

The shared KEK distributed and indicated in glkAlgorithm MUST support the symmetric key-encryption algorithms as specified in [CMSALG] and [CMSAES].

7. Message Transport

SMTP [SMTP] **MUST** be supported. Other transport mechanisms **MAY** also be supported.

8. Security Considerations

As GLOs control setting up and tearing down the GL and rekeying the GL, and can control member additions and deletions, GLOs play an important role in the management of the GL, and only "trusted" GLOs should be used.

If a member is deleted or removed from a closed or a managed GL, the GL needs to be rekeyed. If the GL is not rekeyed after a member is removed or deleted, the member still possesses the group key and will be able to continue to decrypt any messages that can be obtained.

Members who store KEKs **MUST** associate the name of the GLA that distributed the key so that the members can make sure subsequent rekeys are originated from the same entity.

When generating keys, care should be taken to ensure that the key size is not too small and duration too long because attackers will have more time to attack the key. Key size should be selected to adequately protect sensitive business communications.

GLOs and GLAs need to make sure that the generationCounter and duration are not too large. For example, if the GLO indicates that the generationCounter is 14 and the duration is one year, then 14 keys are generated each with a validity period of a year. An attacker will have at least 13 years to attack the final key.

Assume that two or more parties have a shared KEK, and the shared KEK is used to encrypt a second KEK for confidential distribution to those parties. The second KEK might be used to encrypt a third KEK, the third KEK might be used to encrypt a fourth KEK, and so on. If any of the KEKs in such a chain is compromised, all of the subsequent KEKs in the chain **MUST** also be considered compromised.

An attacker can attack the group's shared KEK by attacking one member's copy of the shared KEK or attacking multiple members' copies of the shared KEK. For the attacker, it may be easier to either attack the group member with the weakest security protecting its copy of the shared KEK or attack multiple group members.

An aggregation of the information gathered during the attack(s) may lead to the compromise of the group's shared KEK. Mechanisms to protect the shared KEK should be commensurate with value of the data being protected.

The nonce and signingTime attributes are used to protect against replay attacks. However, these provisions are only helpful if entities maintain state information about the messages they have sent or received for comparison. If sufficient information is not maintained on each exchange, nonces and signingTime are not helpful. Local policy determines the amount and duration of state information that is maintained. Additionally, without a unified time source, there is the possibility of clocks drifting. Local policy determines the acceptable difference between the local time and signingTime, which must compensate for unsynchronized clocks. Implementations **MUST** handle messages with signingTime attributes that indicate they were created in the future.

9. Acknowledgements

Thanks to Russ Housley and Jim Schaad for providing much of the background and review required to write this document.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.
- [CMC] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, June 2008.
- [PROFILE] 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.
- [ACPROF] Farrell, S. and R. Housley, "An Internet Attribute Certificate Profile for Authorization", RFC 3281, April 2002.

- [MSG] Ramsdell, B., Ed., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.
- [ESS] Hoffman, P., Ed., "Enhanced Security Services for S/MIME", RFC 2634, June 1999.
- [CMSALG] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, August 2002.
- [CMSAES] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, July 2003.
- [SMTP] Klensin, J., Ed., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

10.2. Informative References

- [X400TRANS] Hoffman, P. and C. Bonatti, "Transporting Secure/Multipurpose Internet Mail Extensions (S/MIME) Objects in X.400", RFC 3855, July 2004.
- [RANDOM] Eastlake, D., 3rd, Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.
- [FIPS] National Institute of Standards and Technology, FIPS Pub 186-2: Digital Signature Standard, January 2000.

Appendix A. ASN.1 Module

```
SMIMESymmetricKeyDistribution
{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
  pkcs-9(9) smime(16) modules(0) symkeydist(12) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS All --
-- The types and values defined in this module are exported for use
-- in the other ASN.1 modules. Other applications may use them for
-- their own purposes.

IMPORTS

-- PKIX Part 1 - Implicit [PROFILE]
GeneralName
  FROM PKIX1Implicit88 { iso(1) identified-organization(3) dod(6)
    internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
    id-pkix1-implicit(19) }

-- PKIX Part 1 - Explicit [PROFILE]
AlgorithmIdentifier, Certificate
  FROM PKIX1Explicit88 { iso(1) identified-organization(3) dod(6)
    internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
    id-pkix1-explicit(18) }

-- Cryptographic Message Syntax [CMS]
RecipientInfos, KEKIdentifier, CertificateSet
  FROM CryptographicMessageSyntax2004 {iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0)
    cms-2004(24) }

-- Advanced Encryption Standard (AES) with CMS [CMSAES]
id-aes128-wrap
  FROM CMSAesRsaes0aep { iso(1) member-body(2) us(840)
    rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0)
    id-mod-cms-aes(19) }

-- Attribute Certificate Profile [ACPROF]
AttributeCertificate FROM
  PKIXAttributeCertificate { iso(1) identified-organization(3)
    dod(6) internet(1) security(5) mechanisms(5) pkix(7)
    id-mod(0) id-mod-attribute-cert(12) };
```

-- This defines the GL symmetric key distribution object identifier
 -- arc.

```
id-skd OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
  rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) skd(8) }
```

-- This defines the GL Use KEK control attribute.

```
id-skd-glUseKEK OBJECT IDENTIFIER ::= { id-skd 1 }
```

```
GLUseKEK ::= SEQUENCE {
  glInfo          GLInfo,
  glOwnerInfo     SEQUENCE SIZE (1..MAX) OF GLOwnerInfo,
  glAdministration GLAdministration DEFAULT 1,
  glKeyAttributes GLKeyAttributes OPTIONAL }
```

```
GLInfo ::= SEQUENCE {
  glName      GeneralName,
  glAddress   GeneralName }
```

```
GLOwnerInfo ::= SEQUENCE {
  glOwnerName      GeneralName,
  glOwnerAddress   GeneralName,
  certificates     Certificates OPTIONAL }
```

```
GLAdministration ::= INTEGER {
  unmanaged (0),
  managed   (1),
  closed    (2) }
```

```
GLKeyAttributes ::= SEQUENCE {
  rekeyControlledByGLO [0] BOOLEAN DEFAULT FALSE,
  recipientsNotMutuallyAware [1] BOOLEAN DEFAULT TRUE,
  duration [2] INTEGER DEFAULT 0,
  generationCounter [3] INTEGER DEFAULT 2,
  requestedAlgorithm [4] AlgorithmIdentifier
  DEFAULT { id-aes128-wrap } }
```

-- This defines the Delete GL control attribute.
 -- It has the simple type GeneralName.

```
id-skd-glDelete OBJECT IDENTIFIER ::= { id-skd 2 }
```

```
DeleteGL ::= GeneralName
```

-- This defines the Add GL Member control attribute.

```
id-skd-glAddMember OBJECT IDENTIFIER ::= { id-skd 3 }
```

```
GLAddMember ::= SEQUENCE {  
    glName      GeneralName,  
    glMember    GLMember }
```

```
GLMember ::= SEQUENCE {  
    glMemberName      GeneralName,  
    glMemberAddress    GeneralName OPTIONAL,  
    certificates       Certificates OPTIONAL }
```

```
Certificates ::= SEQUENCE {  
    pKC          [0] Certificate OPTIONAL,  
                -- See [PROFILE]  
    aC           [1] SEQUENCE SIZE (1.. MAX) OF  
                AttributeCertificate OPTIONAL,  
                -- See [ACPROF]  
    certPath     [2] CertificateSet OPTIONAL }  
                -- From [CMS]
```

-- This defines the Delete GL Member control attribute.

```
id-skd-glDeleteMember OBJECT IDENTIFIER ::= { id-skd 4 }
```

```
GLDeleteMember ::= SEQUENCE {  
    glName      GeneralName,  
    glMemberToDelete GeneralName }
```

-- This defines the Delete GL Member control attribute.

```
id-skd-glRekey OBJECT IDENTIFIER ::= { id-skd 5 }
```

```
GLRekey ::= SEQUENCE {  
    glName      GeneralName,  
    glAdministration GLAdministration OPTIONAL,  
    glNewKeyAttributes GLNewKeyAttributes OPTIONAL,  
    glRekeyAllGLKeys BOOLEAN OPTIONAL }
```

```
GLNewKeyAttributes ::= SEQUENCE {  
    rekeyControlledByGLO [0] BOOLEAN OPTIONAL,  
    recipientsNotMutuallyAware [1] BOOLEAN OPTIONAL,  
    duration [2] INTEGER OPTIONAL,  
    generationCounter [3] INTEGER OPTIONAL,  
    requestedAlgorithm [4] AlgorithmIdentifier OPTIONAL }
```

-- This defines the Add and Delete GL Owner control attributes.

```
id-skd-glAddOwner OBJECT IDENTIFIER ::= { id-skd 6 }  
id-skd-glRemoveOwner OBJECT IDENTIFIER ::= { id-skd 7 }
```

```
GLOwnerAdministration ::= SEQUENCE {
    glName      GeneralName,
    glOwnerInfo GLOwnerInfo }

-- This defines the GL Key Compromise control attribute.
-- It has the simple type GeneralName.

id-skd-glKeyCompromise OBJECT IDENTIFIER ::= { id-skd 8 }

GLKCompromise ::= GeneralName

-- This defines the GL Key Refresh control attribute.

id-skd-glKeyRefresh OBJECT IDENTIFIER ::= { id-skd 9 }

GLKRefresh ::= SEQUENCE {
    glName      GeneralName,
    dates       SEQUENCE SIZE (1..MAX) OF Date }

Date ::= SEQUENCE {
    start GeneralizedTime,
    end   GeneralizedTime OPTIONAL }

-- This defines the GLA Query Request control attribute.

id-skd-glaQueryRequest OBJECT IDENTIFIER ::= { id-skd 11 }

GLAQueryRequest ::= SEQUENCE {
    glaRequestType  OBJECT IDENTIFIER,
    glaRequestValue ANY DEFINED BY glaRequestType }

-- This defines the GLA Query Response control attribute.

id-skd-glaQueryResponse OBJECT IDENTIFIER ::= { id-skd 12 }

GLAQueryResponse ::= SEQUENCE {
    glaResponseType  OBJECT IDENTIFIER,
    glaResponseValue ANY DEFINED BY glaResponseType }

-- This defines the GLA Request/Response (glaRR) arc for
-- glaRequestType/glaResponseType.

id-cmc-glaRR OBJECT IDENTIFIER ::= { iso(1)
    identified-organization(3) dod(6) internet(1) security(5)
    mechanisms(5) pkix(7) cmc(7) glaRR(99) }
```

-- This defines the Algorithm Request.

id-cmc-gla-skdAlgRequest OBJECT IDENTIFIER ::= { id-cmc-glaRR 1 }

SKDAlgRequest ::= NULL

-- This defines the Algorithm Response.

id-cmc-gla-skdAlgResponse OBJECT IDENTIFIER ::= { id-cmc-glaRR 2 }

-- Note that the response for algorithmSupported request is the
-- smimeCapabilities attribute as defined in MsgSpec [MSG].
-- This defines the control attribute to request an updated
-- certificate to the GLA.

id-skd-glProvideCert OBJECT IDENTIFIER ::= { id-skd 13 }

GLManageCert ::= SEQUENCE {
 glName GeneralName,
 glMember GLMember }

-- This defines the control attribute to return an updated
-- certificate to the GLA. It has the type GLManageCert.

id-skd-glManageCert OBJECT IDENTIFIER ::= { id-skd 14 }

-- This defines the control attribute to distribute the GL shared
-- KEK.

id-skd-glKey OBJECT IDENTIFIER ::= { id-skd 15 }

GLKey ::= SEQUENCE {
 glName GeneralName,
 glIdentifier KEKIdentifier, -- See [CMS]
 glkWrapped RecipientInfos, -- See [CMS]
 glkAlgorithm AlgorithmIdentifier,
 glkNotBefore GeneralizedTime,
 glkNotAfter GeneralizedTime }

-- This defines the CMC error types.

id-cet-skdFailInfo OBJECT IDENTIFIER ::= { iso(1)
 identified-organization(3) dod(6) internet(1) security(5)
 mechanisms(5) pkix(7) cet(15) skdFailInfo(1) }

```
SKDFailInfo ::= INTEGER {  
    unspecified          (0),  
    closedGL             (1),  
    unsupportedDuration  (2),  
    noGLACertificate     (3),  
    invalidCert          (4),  
    unsupportedAlgorithm (5),  
    noGLONameMatch       (6),  
    invalidGLName        (7),  
    nameAlreadyInUse     (8),  
    noSpam               (9),  
    -- obsolete          (10),  
    alreadyAMember       (11),  
    notAMember           (12),  
    alreadyAnOwner       (13),  
    notAnOwner           (14) }  
  
END -- SMIMESymmetricKeyDistribution
```

Author's Address

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

EMail: turners@ieca.com

Full Copyright Statement

Copyright (C) The IETF Trust (2008).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.