IEEE Standard for Interoperable LAN/MAN Security

Clause 3 -- Key Management Protocol

Prepared by the IEEE 802.10 Editor

FOREWORD

(This Foreword is not part of the standard produced by IEEE 802.10.)

IEEE 802.10 was formed in May of 1988 to address the security of Local Area and Metropolitan Area Networks (LANs and MANs). Work on cryptographic key management began in May of 1989. The IEEE 802.10 working group is cosponsored by IEEE 802 and by the Technical Committee on Security and Privacy. IEEE 802.10 provides interoperability standards that are compatible with existing IEEE 802 standards and OSI architectures. The working group is made up of representatives from the vendor, government, and user communities.

Data networks, especially LANs and MANs, have become widespread. LANs and MANs are used by industry and government for transferring vast amounts of data in the course of daily operations. Because of their ever increasing use in the public and private sectors, the capabilities of these networks are being expanded to encompass more and more performance capabilities. As a result, there is a growing need to standardize network protocols whenever feasible to ensure that data networks will interoperate effectively and securely.

As standardization practices evolve, several key areas will become critically important. One of these areas is network security. Depending on the type of work being supported, many LANs and MANs must have the capability to exchange data in a secure manner. This is especially important in cases where disclosure of operational data to unauthorized parties would severely undermine an organization's effectiveness. A related critical area is data reliability. Should data become corrupted, either accidentally or maliciously, the effect on the organization could be grave. Both financial and government institutions have traditionally been aware of the importance of reliable data in a secure environment. However, recent widely publicized cases of computer fraud and related crimes have made this capability a goal for many other industries as well. Security has become a universal concern.

As the need for security on LANs and MANs becomes more widely recognized, the need for a standardized approach to providing such a capability also becomes a priority. Much security standardization has already started. Where applicable, this standard incorporates this previous work.

COMMITTEE LIST

Members of the LAN/MAN Security Working Group who participated in the development of this standard are: [Filled in Later]

1. INTRODUCTION

1.1. Scope and Purpose

This document specifies a cryptographic key management model and protocol. The Key Management Protocol is an OSI Basic Reference Model Application Layer protocol. It uses the services provided by the Security Exchange Service Element (SESE), the Association Control Service Element (ACSE) and the Presentation service. In order to retrieve presentation addresses or to support a specific key management scheme the Directory service may be needed. A key management protocol entity provides a service within the context of a defined security policy; it is outside the scope of this document to define such security policies.

The key management protocol entity supports security protocol entities by managing the cryptographic keying material that is necessary for the provision of secure communication. Key management protocol entities are responsible for creating the cooperative relationship and establishing the required cryptographic keying material. Security protocols supported by this key management protocol include the Security Data Exchange Security Protocol, IEEE 802.10b; the Network Layer Security Protocol, ISO 11577; and the Transport Layer Security Protocol, ISO 10736. This Key Management Protocol is intended to support other security protocols, including application layer security protocols, which are stll under development.

The key management model and protocol support three key management techniques: manual key distribution, center-based key distribution, and certificate-based key distribution. Symmetric cryptographic algorithms are normally used with center-based key distribution techniques. Asymmetric cryptographic algorithms are always used with certificate-based key distribution techniques.

[expand as necessary]

1.2. Overview

The Key Management Protocol is used to create, spawn, and delete the security associations that are needed by security protocols. The security association is composed of a symmetric keying material and security attributes. The security attributes determine how the security protocol will use the symmetric keying material.

This standard uses the functions and services defined in Generic Upper Layers Security (GULS), ISO 11586, to provide an application-level communications infrastructure for the development of a mechanism-independent Key Management Protocol.

GULS defines defines a security exchange (SE) mechanism that permits the exchange of security information such as keying material in support of the various security services. The Security Exchange Service Element (SESE), an application level element, uses ASN.1 notational tools to allow the consistent specification of SEs carried in the SESE. GULS also defines security transformations (STs) to provide integrity, confidentiality, or integrity and confidentiality of application data communicated between open systems. STs consist of:

_ a transfer-syntax function, the Generic Protecting Transfer Syntax, allowing applications to signal their integrity and confidentiality requirements in communication protocol;

_ ASN.1 notational tools to allow the consistent specification of STs, as well as a mechanism allowing an application to map its integrity and confidentiality requirements to specific security transformations which satisfy these requirements.

This standard uses the functions defined in GULS to define SEs and STs that are applicable to the Key Management Techniques discussed in Clause 5. This standard also defines specific uses of the SESE, and application-contexts that define specific procedural rules and constraints pertaining to the overall use of the GULS functions in support of Key Management.

[expand as necessary]

2. DEFINITIONS

2.1 Acronyms

ACSE	Association Control Service Element
ASO	Application Service Object
CF	Control Function
CKD	Center for Key Distribution
CKT	Center for Key Translation
GULS	Generic Upper Layer Security
KCASO	Key Center Application Service Object
KMAE	Key Management Application Entity
KMAP	Key Management Application Process
KPASO	Key Peer Application Service Object
SESE	Security Exchange Service Element
SMIB	Security Management Information Base

[expand as necessary]

2.2 Security Definitions

[TBD]

3. REFERENCES

IS 7498, Information Processing Systems - Open Systems Interconnection - Basic Reference Model, October 15, 1984.

IS 7498-2, Security Architecture

IS 7498-3

Secure Data Exchange (?)

DIS 11586, Generic Upper Layer Security

IS 8649, ACSE Service Definition

IS 8650, ACSE Protocol Specification

DIS 9545

IS 10736, Transport Layer Security Protocol

IS 11577, Network Layer Security Protocol

IS 9594-8, Directory Authentication Framework (.a.k.a. X.509)

[expand as necesary]

4. KEY MANAGEMENT TECHNIQUES

This section describes the key management techniques that are supported by this protocol. Three classes of key management techniques are supported: manual distribution, center-based key distribution, and certificate-based distribution. The key center distribution techniques support both Centers for Key Distribution (CKDs) and Centers for Key Translation (CKTs). The format of certificates used in the certificate-based distribution techniques is described in CCITT Recommendation X.509.

[Specific examples of each technique will be in an informative appendix.]

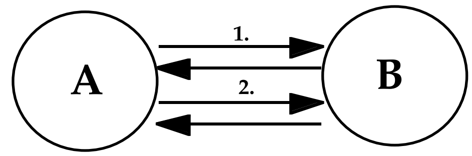
4.1. Manual Key Distribution Techniques

Manual key distribution techniques may be used to establish pairwise or multicast cryptographic keying material.

These methods are based on off-line delivery of secret keys to intended users. Manual methods of secret key distribution may be cumbersome and have scalability problems. However, in many cases manual key delivery is required only once per user, while the later key distribution activities can be performed remotely, using the previous value of the key as a key encryption key (KEK) to cover each new key, or key package as a whole, using the KEK. The methods/procedures for putting preplaced key material in appropriate holding areas to be used by KMP are beyond the scope of KMP. KMP provides a means by which preplaced keys can be used in the establishment and management of security associations. Preplaced keys are suitable for multicast key distribution, and are often the preferred means of distributing "group" or "net" keys.

Manual key distribution methods rely on off-line and ad hoc methods of user authentication. Peer, group, and net entity authentication is derived from the liveness (successful use) of the key, and is totally dependent on the strength of procedures used to distribute the key manually.

Manual key distribution may be used in an environment, when two entities already share a key that can be used to establish further keys. This method requires the use of time variant parameters, such as time stamps, sequence numbers or random numbers. The term random numbers also includes unpredictable pseudo-random numbers. Including time variant parameters plus identification information supports the mutual authentication of the parties communicating.



1. A and B select a key that was previously distributed manually.
2. A and B use the key to protect the negotiation of the attributes for the operation of the security protocol.

Figure 3-1 -- Selecting a manually distributed key

4.2. Center-based Key Distribution Techniques

Center-based key distribution techniques may be used to establish pairwise or multicast cryptographic keying material.

Keying material established using the technique described by Needhan and Schroeder can be based on either a Key Distribution Center(CKD) or a Key Translation Center (CKT). Center based key distribution covers both techniques. The protocols depend upon the manual distribution of key encrypting keys (KKs) or key encrypting key pairs (*KKs) using dual control, split knowledge procedures to provide confidentiality and integrity protection for those keys. The terms dual control and split knowledge mean that the key is composed of two key components which will later be combined to form the cryptographic key. Each component conveys no knowledge of the resultant key. Once the manual keys have been distributed further *KKs and KDs(data keys) may be electronically distributed. Confidentiality for the automatically distributed keys is provided by encryption based on the manually distributed keys. Integrity may also be provided by the presence of an Integrity Check Value (ICV).

The following is an example of the protocol exchanges required in the center environment.

Each principle has a secret key that is known only to itself and the center. the contents of which are secret. The main step in setting up secure communications between A and B is for the initiator, A, to generate a message with two properties: 1) It must be comprehensible only to B, i.e. allow only B to use its contents to identify itself to A. 2) It must be evident to B that it originated with A.

The protocol starts with A communicating in the clear to the KDC his own claimed identity and the identity of the desired correspondent, B, together with A's unique identifier for this transaction, IA1. The identifier must be different than any other used in previous messages of the same type that were sent by A. The first message of the protocol is:

A => KDC: A,B,IA1 1.1

The KDC upon receiving 1.1 looks up the secret , identifying keys of both A and B and also computes a new key CK that will be the key used for the secure conversation between A and B. The KDC sends back to A the following message:

KDC => A {IA1,B,CK,{CK,A}KB}KA 1.2

where KA and KB are A's and B's secret key. Since 1.2 is encrypted with A's secret key, only A can decrypt it and discover the conversation key CK. Following decryption, A checks for the presence of the intended recipient's name,B, and the correct unique identifier, IA1, in order to verify that the message actually is a reply by the KDC to the current inquiry. Both the name of the intended recipient and the unique identifier must appear in message 1.2. If the recipient's name is left out, then an intruder could change that name in the first message (1.1), say to C, before KDC receives it, with subsequent result that A would unknowingly communicate with C instead of B. If the unique identifier is left out, then an intruder could substitute a previously recorded message 1.2 and force A to reuse a previous conversation key. A retains the CK and sends the part encrypted with KB to B:

A B: {CK,A}KB 1.3

The real B, and no other impostor, will be able to decrypt the third message and also emerge with the conversation key CK, the same as A has. B also knows the identity of A as authenticated by the KDC.

A now knows that any communication he receives encrypted with CK must have originated with B, and also that any communication A sends with CK encryption will be understood only by B. Both are known because the only messages containing CK that have ever bee sent are tied to A's and B's secret keys. B is in a similar state. There is the one concern that no part of the protocol exchanges or ensuing conversation are being replayed by an intruder from a recording of a previous conversation between A and B. A is aware that he has not used the key CK before and therefore has no reason to believe that messages encrypted with it is other that the legitimate response from B. Unless B can recall indefinitely keys previously used with A he is not sure that the third message and any subsequent messages from A are not being replayed. To insure that this not the case, B generates a unique identifier for the transaction IB and sends it to A encrypted under the CK key:

B => A: {IB}CK 1.4

expecting a related reply , minus one:

A => B: {IB-1}CK 1.5

If this reply message is satisfactorily received, then the mutual confidence is sufficient to enable substantive communication, encrypted with CK, to begin.

In the above scenario five messages were exchanged. This number may be reduced to three by A keeping, a cache of items of the form B:CK,{CK,A}KBderived from the message 1.2 thus eliminating the first two messages. With caching, the same CK is being sent again and again, so the unique identifier handshakes need to be two-way by replacing steps 1.3 and 1.4 with:

A B: {CK,A}KB, {IA2}CK 1.3'

B A: {IA2-1,IB}CK 1.4'

A bubble diagram of the above exchange follows:

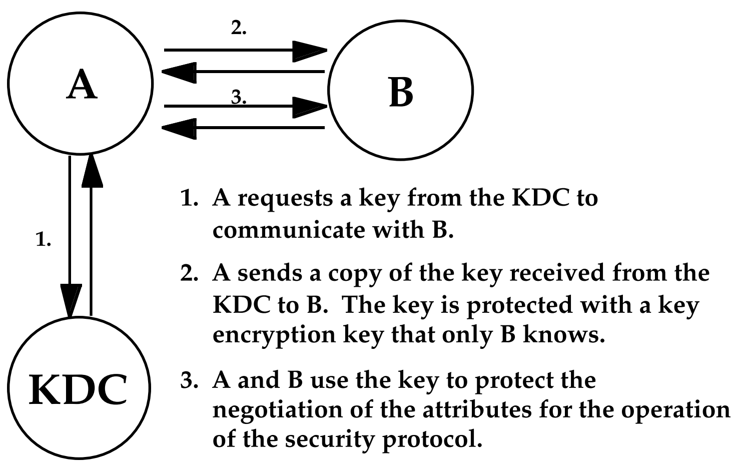


Figure 3-2 -- Center-based key distribution

4.3. Certificate-based Distribution Techniques

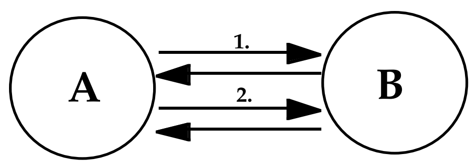
Certificate-based key distribution techniques may be used to establish pairwise cryptographic keying material, but certificate-based key distribution techniques may not be used to establish multicast cryptographic keying material.

Certificate-based distribution techniques employs public key cryptographic techniques, which come in two flavors.

- Key Distribution: A public key cryptographic key technique that protects the distribution of a symmetric cryptographic key generated by the local key management system while it is sent to the remote key management end system.

- Key Agreement: A public key cryptographic technique that computes a common symmetric cryptographic key at the local and remote key management end system.

This standard uses the format for certificates defined in the Directory Authentication Framework, CCITT Recommendation X.509. This technique uses public key cryptography, and the X.509 format certificates contain the public component of the asymmetric key pair. Of course, the private component of the asymmetric key pair must be kept confidential. The certificate also contains the identity of the certificate owner; this identity takes the form of a distinguished name. Certificates are digitally signed by a recognized certification authorities. By signing the certificate, the certification authority is vouching for the binding of the public component of the asymmetric key pair and the identity. That is, the entity named by the distinguished name holds the private component of the asymmetric key pair. This knowledge is used in authentication.



1. A and B exchange key generation information, which they use to make a secret key.
2. A and B use the key to protect the negotiation of the attributes for the operation of the security protocol.

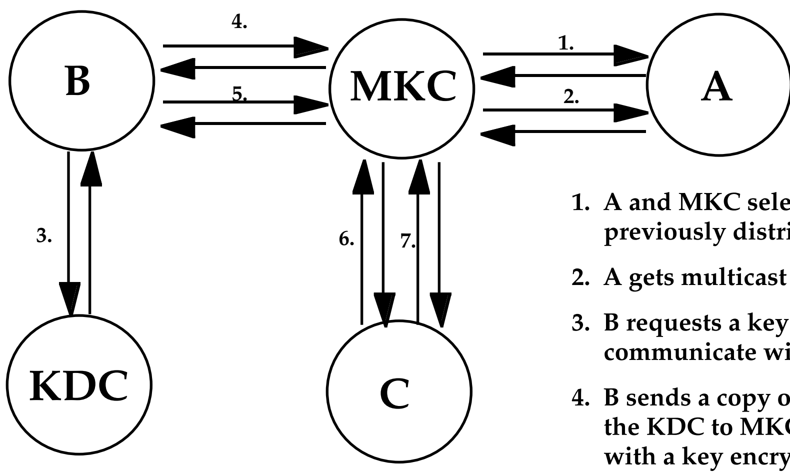
Figure 3-3 -- Certificate-based key distribution

4.4. Multicast Key Distribution Techniques

Multicast key distribution relies on one of the other key distribution techniques, manual, center-based or certificate-based, to establish a security association between the local key management end system and the Multicast Key Center (MKC). The security association must provide peer entity authentication and confidentiality; it may also provide integrity. Under the protection of the security association, the local key management system requests the broadcast or multicast keys and their attributes from the multicast key center. The multicast key center determines if the local key management system is authorized to receive the multicast keys that were requested; if it is authorized, the multicast keys are sent to the local key management system.

Each security protocol entity instigates obtaining the broadcast or multicast key as its needs dictates. The key center does not push the broadcast or multicast keys out to every member of the group.

The security association identifier and all other security association attributes are assigned by the multicast key center so that the same ones will be used by all Security Protocol entities who posses the multicast key.



1. A and MKC select a key that was previously distributed manually.

2. A gets multicast key from MKC.

3. B requests a key from the KDC to communicate with MKC.

4. B sends a copy of the key received from the KDC to MKC. The key is protected with a key encryption key that only MKC knows.

5. B gets multicast key from MKC.

6. A and B exchange key generation information, which they use to make a secret key.

7. B gets multicast key from MKC.

Figure 3-4 -- Multicast key distribution

5. KEY MANAGEMENT MODEL

A security association is cryptographic keying material coupled with security attributes, and security associations are represented as management objects in the Security Management Information Base (SMIB). The SMIB is the set of all security-related management objects, and it is conveyed across all layers of the OSI Reference Model. Thus, creating a security association establishes cryptographic keying material and attributes within the SMIB; spawning a security association uses an existing security association to create another one within the SMIB; and deleting a security association removes the cryptographic keying material and attributes from the SMIB. The Key Management Protocol is used to create, spawn, and delete security associations within the SMIBs of communication end systems.

5.1 Security Association Lifecycle

Security associations are created by coupling cryptographic keying material to security attributes. This is accomplished in two phases: establishing the cryptographic keying material, and negotiating the security attributes. Once the cryptographic key is established, it is used to protect the security attribute negotiation.

The negotiated security attributes determine how the security association will be used. In other words, the security attributes determine which Security Protocol will use the security association and how that Security Protocol will use the cryptographic keying material.

After a security association is created, it may be used to spawn other security associations. The new security association may have the same or different attributes as the parent security association, but the Security Association Identifier (SAID) will always be different. The parent security association remains unaffected spawning another security association. The cryptographic keying material from the parent security association is transformed to generate the new cryptographic keying material for the new security association

When the security association is no longer needed, it is deleted. Either key management end system can determine that the security association is no longer needed, then that key management end system notifies the remote key management end system that the security association is being deleted.

5.2 Key Management Application Entity Structure

The key management application requires communication with remote key management applications. To support this communication, the Key Management Protocol relies on OSI communication services provided by the

Association Control Service Element (ACSE) and the Security Exchange Service Element (SESE). These service elements interface to the OSI Presentation Layer to communicate with the remote key management end system.

The Key Management Application Process (KMAP) is not specified by this standard, but the KMAP uses the OSI communications services to establish cryptographic keying material, negotiate attributes for the cryptographic keying material, and delete the cryptographic keying material in accordance with locally-defined security policy. The KMAP provides the top level interface to the user. The KMAP may use one or more of the supported cryptographic keying material distribution techniques: manual key distribution, center-based key distribution, and certificate-based key distribution. A single attribute negotioation technique is used with all three cryptographic keying material distribution techniques. This protocol is not used to establish security associations that provide no security services.

The Key Management Application Entity (KMAE) is an Application Service Object (ASO) that provides the OSI communication services necessary to perform cryptographic keying material management. As shown in Figure 1, the KMAE comprises two application service objects and a control function. The Key Peer Application Service Object (KPASO) interacts with the remote key management end system to establish security associations. The Key Center Application Service Object (KCASO) interacts with a Center for Key Distribution (CKD) or a Center for Key Translation (CKT) to obtain cryptographic keying material. The Control Function (CF) coordinates interactions between the KPASO and the KCASO. Section 6.1 describes the service interface to the KMAE.

The KPASO comprises the Association Control Service Element (ACSE), the Security Exchange Service Element (SESE), and a Control Function (CF). The ACSE is a common application service element defined by IS 8649, and it establishes and terminates application associations. The SESE is a common application service element defined by DIS 11586, and it performs security exchanges and security transformations. The CF coordinates interactions between ACSE and SESE. Section 6.2 describes the service interface to the KPASO.

The KMAE may invoke the KPASO multiple times, once for each remote key management end system. Therefore, each KPASO invocation uses a single application association. Limiting the KPASO to one application association greatly simplifies addressing.

The KCASO, like the KPASO, comprises the ACSE, the SESE, and a CF. Again, The CF coordinates interactions between ACSE and SESE. Section 6.3 describes the service interface to the KCASO.

The KMAE may invoke the KCASO multiple times, once for each CKD or CKT.

Therefore, each KCASO invocation uses a single application association. Again, limiting the KCASO to one application association greatly simplifies addressing.

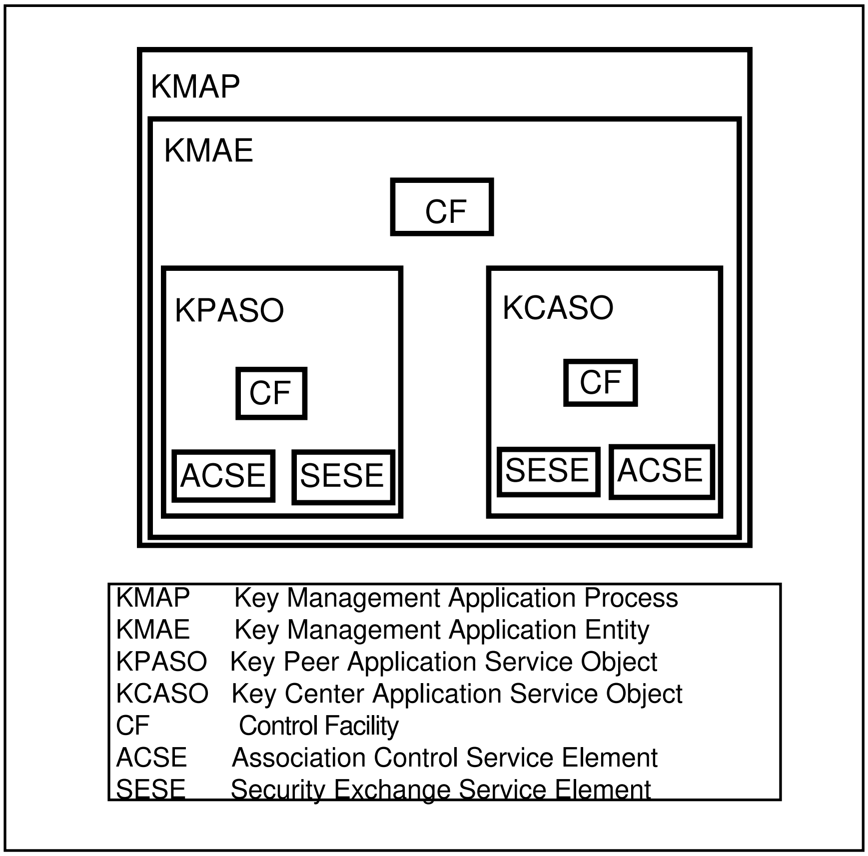


Figure 3-5 -- Key management application structure

5.3 Sequencing of Application Layer Services

The KMAE does not implement any security exchanges, rather the KMAE invokes the KPASO and KCASO to perfom the necessary security exchanges. While the communications between the KMAE and the KPASO and the communications between the KMAE and the KCASO are not subject to standardization, they are controlled by the Control Function (CF) within the KMAE; these communications are a local matter.

The KMAE may optionally use the KPASO to negotiate the key management technique and the key management algorithm that will be used to establish symmetric cryptographic keying material. Regardless of how the symmetric key material is established, the KMAE always uses the KPASO to establish the symmetric key material, and then to perform the attribute negotiation exchange to complete the creation of the security association.

When using center-based techniques the KMAE uses the KCASO to obtain symmetric cryptographic keying material from a CKD, or to translate a key using a CKT. This symmetric cryptographic keying material is subsquently used by the KPASO to complete key establishment.

5.3.1 Manually Distributed Key

To select a manually distributed key, a Create-SA service is issued to the KMAE. If the Create-SA service arguments include the Key Management Technique Identifier List, then the KMAE CF takes the Key Technique Algorithm Identifier List and invokes the KPASO service Pick-KM-ALG. The KPASO generates the proper SEIs from the passed arguments, then passes the SEIs to SESE (via the SE-Transfer service). The KPASO CF passes the SEIs to SESE only after an application-association is in place for the SESE to use. If an application-association is not already in place, the KPASO CF invokes the A-Associate service provided by ACSE. Once the KMAE CF has selected the manual key distribution technique, the KMAE CF issues the Select-Key service to the KPASO. Then the KMAE CF uses the Security Protocol Parameters List to issue the Pick-SA-Attrs. If further key management communications are likely, the KMAE CF leaves application-association in place; otherwise the KMAE CF releases the application-association.

The following figure is for illustrative purposes only. It does show the relationship between the KMAE and the KPASO. As mentioned above, the Release-P service is optional, but has been included in the figure for completeness. Only service requests are shown; confirmations have been omitted to simplify the figure.

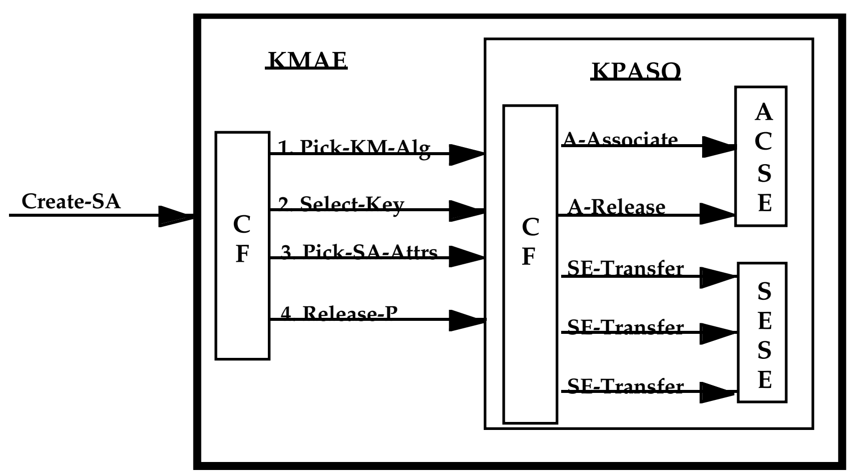


Figure 3-6 -- Create SA by selecting manually distributed key

5.3.2 Key Center Distribution

To obtain a key from a key distribution center or to translate a key by a key translation center, a Create-SA service is issued to the KMAE. If the Create-SA service arguments include the Key Management Technique Identifier List, then the KMAE CF takes the Key Management Technique Algorithm Identifier List and invokes the KPASO service Pick-KM-ALG. The KPASO generates the proper SEIs from the passed arguments, then passes the SEIs to SESE (via the SE-Transfer service). The KPASO CF passes the SEIs to SESE only after an application-association is in place for the SESE to use with its peer in the remote key management end system. If an application-association is not already in place, the KPASO CF invokes the A-Associate service provided by ACSE. Once the KMAE CF has selected the key center or translation center distribution technique, the KMAE CF issues the REQUEST-KEY service to the KCASO. In the same manner as the KPASO, the KCASO CF passes the SEIs to SESE only after an application-association is in place for the SESE to use with its peer in the remote key center end system. If an application-association with the key distribution center or key translation center does not already exist, the KCASO CF invokes the A-Associate service provided by ACSE. After the KMAE CF receives confirmation from the REQUEST-KEY service, the KMAE CF issues the SEND-KEY service to the KPASO. Then the KMAE CF uses the Security Protocol Parameters List to issue the Pick-SA-Attrs. If further key management communications are likely, the KMAE CF leaves both application-associations in place; otherwise the KMAE CF releases the application-associations.

The following figure is for illustrative purposes only. It does show the relationship between the KMAE, the KPASO and KCASO. As mentioned above, the Release-P and Release-C services are optional, but have been included in the figure for completeness. Only service requests are shown; confirmations have been omitted to simplify the figure.

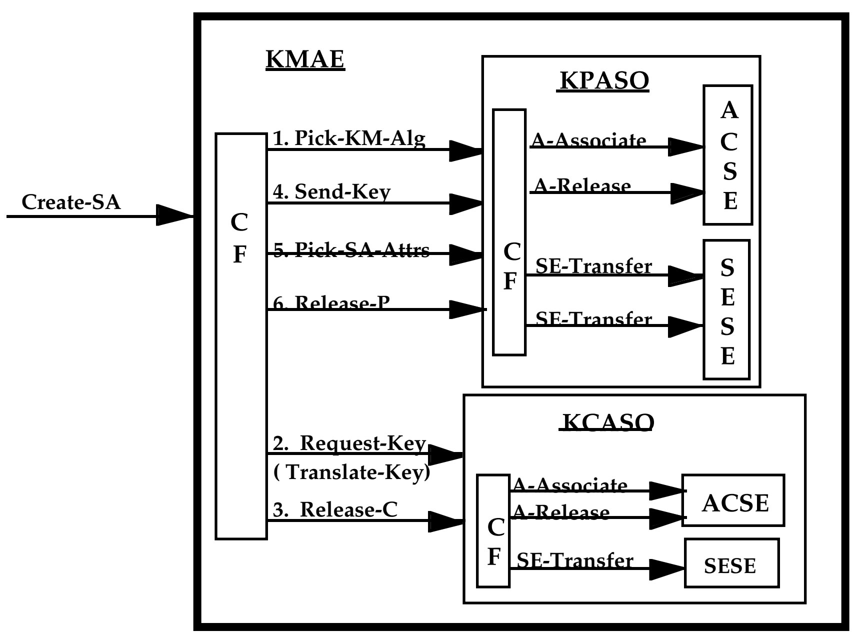


Figure 3-7 -- Creat SA using center-based key distribution

5.3.3 Certificate-Based Key Distribution

To generate a key using certificate based techniques, a Create-SA service is issued to the KMAE. If the Create-SA service arguments include the Key Management Technique Identifier List, then the KMAE CF takes the Key Technique Algorithm Identifier List and invokes the KPASO service Pick-KM-ALG. The KPASO generates the proper SEIs from the passed arguments, then passes the SEIs to SESE (via the SE-Transfer service). The KPASO CF passes the SEIs to SESE only after an application-association is in place for the SESE to use. If an application-association is not already in place, the KPASO CF invokes the A-Associate service provided by ACSE. Once the KMAE CF has selected the certificate based key distribution technique, the KMAE CF issues the Make-Key service to the KPASO. Then the KMAE CF uses the Security Protocol Parameters List to issue the Pick-SA-Attrs. If further key management communications are likely, the KMAE CF leaves application-association in place; otherwise the KMAE CF releases the application-association.

The following figure is for illustrative purposes only. It does show the relationship between the KMAE and the KPASO. As mentioned above, the Release-P service is optional, but has been included in the figure for completeness. Only service requests are shown; confirmations have been omitted to simplify the figure.

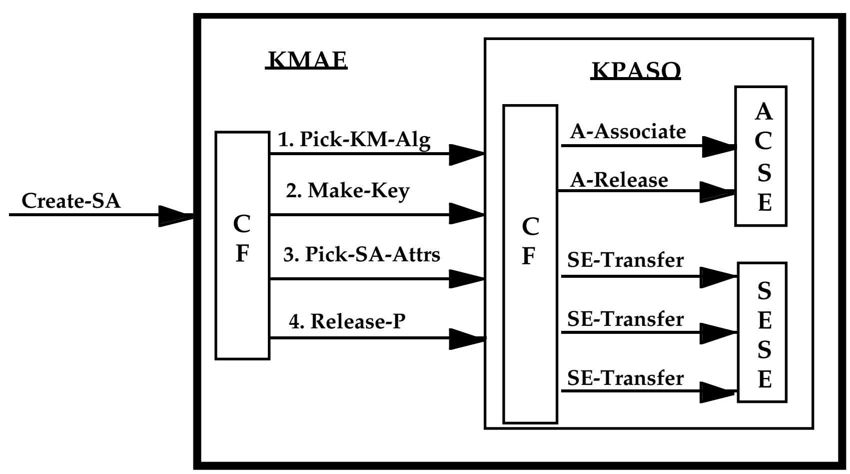


Figure 3-8 -- Create SA using certificate-based key distribution

5.3.4 Multi-Cast Key Distribution

For Multi-Cast keys, a Create-SA service is issued to the KMAE. The KMAE CF takes information from the Key Management Algorithm Identifier List and issues the KPASO service Pick-KM-Alg. The KPASO generates the proper SEI's from the passed parameters, and then passes the SEIs to SESE (via the SE-Transfer service). The KPASO CF passes the SEIs to SESE only after an application-association is in place for the SESE to use with its peer in the remote multi-cast key center. If an application-association is not already in place, the KPASO CF invokes the A-Associate service provided by ACSE. Once the KMAE CF has selected the key distribution technique, the KMAE CF issues either a SELECT-KEY, MAKE-KEY, or SEND-KEY service to the KPASO. After the KMAE CF has received confirmation from the KPASO for the chosen service, the KMAE CF issues the GET-MKEY to the KPASO. If further key management communications are likely the application-association is left in place; otherwise it is released.

The following figure is for illustrative purposes only. It does show the relationship between the KMAE and the KPASO. As mentioned above, the Release-P service is optional, but has been included in the figure for completeness. Only service requests are shown; confirmations have been omitted to simplify the figure.

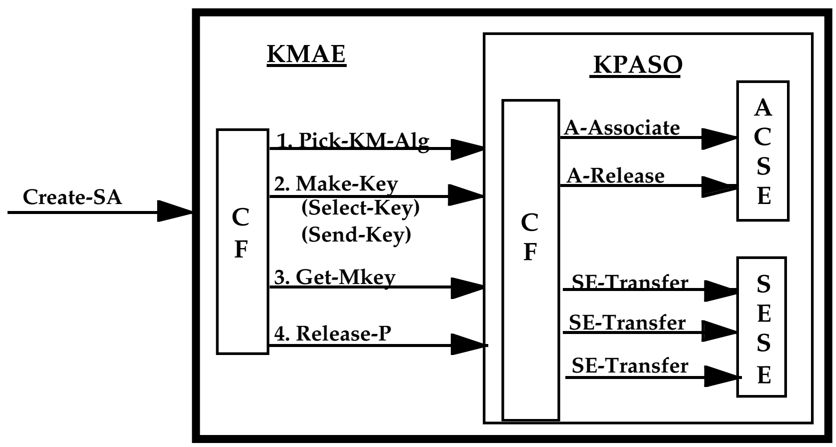


Figure 3-9 -- Multicast key distribution

5.3.5 SPAWN Security Association

To generate a new Security Association from an existing one, by either changing the attributes while preserving the key material, transforming the key material while preserving the attributes, or both, a Spawn-SA service is issued to the KMAE. The KMAE CF takes information from the Key Transformation Algorithm Identifier List and the Previously Established Calling and Called SAIDs and invokes the KPASO service Spawn-Key. The KPASO generates the proper SEIs from the passed parameters, then passes the SEI's to the SESE (via the SE-Transfer service). The KPASO CF passes the SEIs to SESE only after an application-association is in place for the SESE to use with its peer in the remote key management end system. If an application-association is not already in place, the KPASO CF invokes the A-Associate service provided by ACSE. The KMAE CF uses the Security Protocol Parameters List to issue the Pick-SA-Attrs to the KPASO. If further key management communications are likely the application-association is left in place; otherwise it is released.

The following figure is for illustrative purposes only. It does show the relationship between the KMAE and the KPASO. As mentioned above, the Release-P service is optional, but has been included in the figure for completeness. Only service requests are shown; confirmations have been omitted to simplify the figure.

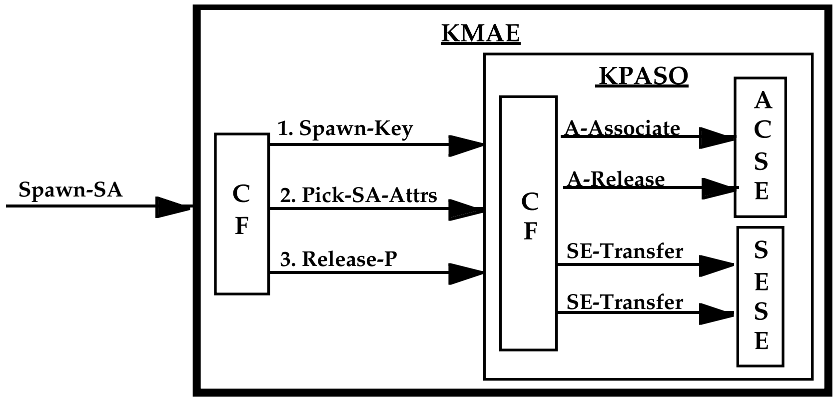


Figure 3-10 -- Spawn a SA

5.3.6 Delete Security Association

To delete a security association, a Delete-SA service is issued to the KMAE. The KMAE CF uses the Calling and Called SAIDs to issue the Delete-Key service to the KPASO. The KPASO generates the proper SEIs from the passed parameters, then passes the SEI's to the SESE (via the SE-Transfer service). The KPASO CF passes the SEIs to SESE only after an application-association is in place for the SESE to use with its peer in the remote key management end system. If an application-association is not already in place, the KPASO CF invokes the A-Associate service provided by ACSE. If further key management communications are likely the application-association is left in place; otherwise it is released.

The following figure is for illustrative purposes only. It does show the relationship between the KMAE and the KPASO. As mentioned above, the Release-P service is optional, but has been included in the figure for completeness. Only service requests are shown; confirmations have been omitted to simplify the figure.

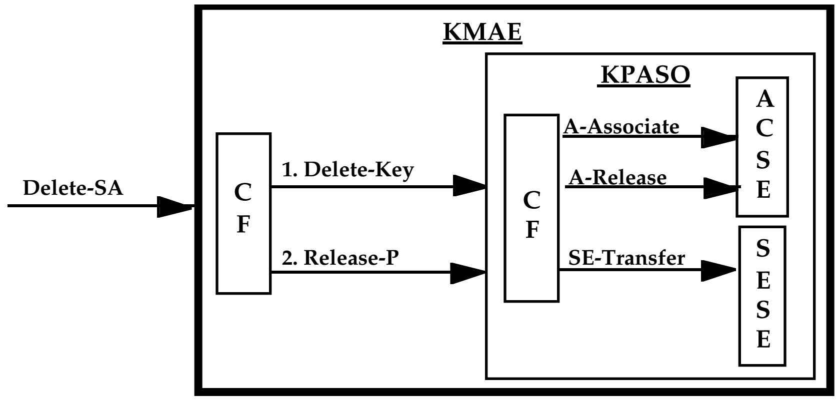


Figure 3-11 -- Delete a SA

6. SERVICE DEFINITION

The KMAE provides services to the KMAP using a serviceinterface with a very high level of abstraction. The KPASO and KCASO provideservice to the KMAE using a service interface with a lower level ofabstraction. The KMAE CF determines when application associations should be established and released. Thetime at which application associations should be released is not addressed bythis standard; it is a local matter.

6.1. Key Management Application Entity (KMAE) ServiceDefinition

The KMAE provides three services: CREATE-SA, SPAWN-SA, andDELETE-SA.

6.1.1. Create Security Association (CREATE-SA)

The CREATE-SA primitive establishes the cryptographic keying material and associated attributes to form a security association. CREATE-SA provides a confirmed service, and it securely establishes the same symmetric cryptographic keying material in the SMIB of each key management end system. The specific attributes that are associated with the cryptographic key depends on the Security Protocol that will use the security association. The enabling of security associations within the SMIB for use by the Security Protocol is a local matter; however, when the CREATE-SA.confirm is provided, the security association has been enabled for use by the Security Protocol on both of the peer key management end systems.

The parameters of CREATE-SA defined in the following table:

CREATE-SA Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)		
Called AE-Title	M	M (=)	M	M (=)
Key Management Technique Identifier List	U	C (=)		
Key Management Technique Identifier			U	C (=)
Security Protocol Parameters List	M	M (=)		
Security Protocol Parameters			M	M (=)
Calling SAID		M	M (=)	M (=)
Called SAID			M	M (=)
Result			M	M (=)

The Security Protocol Parameters may be listed explicity or they may be implicitly listed by naming an existing security association.

[Insert a discussion of each parameter.]

6.1.2. Spawn Security Association (SPAWN-SA)

The SPAWN-SA service primitive uses a previously established security association to establish a new security association. The new security association may have the same or different attributes as the previously established security association. If the attributes are the same, the security association identifier attribute will be different for the two security associations. The previously established security association remains uneffected by the SPAWN-SA service primitive. SPAWN-SA provides a confirmed service. The cryptographic keying material from the previously established security association is transformed to generate the new cryptographic keying material for the second security association. The same transformation is performed by both key management end systems to generate the same symmetric cryptographic keying material; this cryptographic keying material is established in the SMIB of each key management end system. The enabling of cryptographic keying material and associated attributes within the SMIB for use by the Security Protocol is a local matter; however, when the SPAWN-SA.confirm is provided, the cryptographic keying material and associated attributes have been enabled for use by the Security Protocols on both of the peer key management end systems.

The SPAWN-SA exchange is protected by the previously established security association. This protection provides peer entity authentication; the protection also provides confidentiality, integrity, or confidentiality and integrity.

The parameters of SPAWN-SA defined in the following table:

SPAWN-SA Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)		
Called AE-Title	M	M (=)	M	M (=)
Key Transformation Algorithm Identifier	M	M (=)		
Previously Established Calling SAID	M	M (=)		
Previously Established Called SAID	M	M (=)		
Calling SAID		M	M (=)	M (=)
Called SAID			M	M (=)
Result			M	M (=)

The Key Transformation Algorithm Identifier is selected from the set of transformation algorithms negotiated when the previous security association was created. If mutliple keys exist in the previously established security association, the identifier tells how each key is to be transformed.

[Insert a discussion of each parameter.]

6.1.3. Delete Security Association (DELETE-SA)

The DELETE-SA primitive notifies the remote key management end system that the cryptographic keying material and associated attributes which define a particular security association are being deleted from the local SMIB. DELETE-SA provides an unconfirmed service. The remote key management end system also deletes the cryptographic keying material and associated attributes from the remote SMIB.

The DELETE-SA exchange is protected by the previously established security association. This protection provides data origin authentication; the protection also provides confidentiality, integrity, or confidentiality and integrity.

The parameters of DELETE-SA defined in the following table:

DELETE-SA Parameter Name	Req	Ind
Calling AE-Title	M	M (=)
Called AE-Title	M	M (=)
Calling SAID	M	M (=)
Called SAID	M	M (=)

[Insert a discussion of each parameter.]

[Calling SAID could be Optional.]

6.2. Key Peer Application Service Object (KPASO) Services

The KPASO provides ten services: PICK-KM-ALG, SELECT-KEY, MAKE-KEY, SEND-KEY, PICK-SA-ATTRS, SPAWN-KEY, GET-MKEY, DELETE-KEY, RELEASE-P, and ABORT-P.

6.2.1. Negotiate Key Management Algorithm (PICK-KM-ALG)

The PICK-KM-ALG primitive choses the key management algorithm that will be used to establish the same cryptographic symmetric key in the two key managemet end systems. Use of this primitive is optional. In many cases, the calling key management end systems will support a single key management algorithm, thus selection is unnecessary. PICK-KM-ALG provides a confirmed service. If there is not an application association with the remote key management end system, then one is created.

Since this PICK-KM-ALG exchange is unprotected, any security association attributes that are selected, either implicitly or explicitly, in this exchange should be revalidated in the protected PICK-SA-ATTRS exchange.

The parameters of PICK-KM-ALG are defined in the following table:

PICK-KM-ALG Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)		
Called AE-Title	M	M (=)	M	M (=)
Key Management Technique Identifier List	M	M (=)		
Key Management Technique Identifier			M	M (=)
Security Domain Identifier List	U	C (=)		
Security Domain Identifier			U	C (=)
Result			M	M (=)

[Insert a discussion of each parameter.]

[Changed Key Management Algorithm Identifier to Key Management Technique Identifier so as not to confuse this with AlgorithmIdentifier from X.509. We may need to revise this if Technique Identifers are not different than Algorithm Identifiers]

6.2.1.1 Security Domain Identifier

[change parameter name from Security Domain Id]
This optional parameter is used by the calling party to identify a specific security policy (identifying a set of pre-established rules) to the called party in the security association establishment. The parameter can be conveyed in one of two forms: 1) a clear text object identifier, alone; or, 2) a clear text object identifier, followed by authentication information. The purpose of this optional parameter is to agree upon the basis of secure communications before going through the key management exchanges. The object identifier representing the security policy identifying information signifies to the called party how to use the optional authentication information, based on prior agreement.

This optional parameter has many potential use scenarios. In the first scenario it is used to signify basic agreement to form a security association before actually commiting to the exhange to put key in place. In the second scenario it is used to signify basic aggreement to form a security association before committing to the exhange to put key in place, but with greater assurrance of the identity of the calling party to the called party. In the third scenario, it is used to signify basic agreement to form a security associaton between communicating entities carried over a security association which has already been established between security managers. Here all of the information exchanged during the establishment of the (2nd) security association is covered by a security protocol which had its security association put in place between the security managers before engaging in this (2nd) security association establishment activity.

It is properly noted that the first and second scenarios are vulnerable to a denial of service attack; i.e., by an intrusive listener modifying the content of the security policy identifier information. Nevertheless, this is a security policy matter, and in some environments it may be permitted. The denial of service vulnerability does not exist in the third scenario, and there are known and well established requirements to support this kind of scenario in certain user communities.

6.2.2. Select Key (SELECT-KEY)

The SELECT-KEY primitive chooses a symmetric cryptographic key from a cache of manually distributed keys that are stored in the SMIB. The key selected must not be presently assigned to a security association. SELECT-KEY provides a confirmed service. If there is not an application association with the remote key management end system, then one is created.

The parameters of SELECT-KEY are defined in the following table:

SELECT-KEY Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)		
Called AE-Title	M	M (=)	M	M (=)
Cryptographic Keying Material Identifier	M	M (=)		
Calling SAID	M	M (=)	M (=)	M (=)
Called SAID			M	M (=)
Result			M	M (=)

[Insert a discussion of each parameter.]

The Cryptographic Keying Material Identifier names a pairwise key from the cache.

[Do we need to add SELECT-MKEY to suport manually distributed multicast keys? Or do expand the context of SELECT-KEY to deal with pairwise and multicast keys?]

6.2.3. Make Key (MAKE-KEY)

The MAKE-KEY primitive obtains the data needed to establish a symmetric cryptographic key using a public key cryptographic technique. The public key cryptographic key technique may be a key distribution technique or a key agreement technique. The public keys are bound to the distinguished name of the key management end system in X.509 certificates. MAKE-KEY provides a confirmed service. If there is not an application association with the remote key management end system, then one is created.

The parameters of MAKE-KEY are defined in the following table:

MAKE-KEY Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)		
Called AE-Title	M	M (=)	M	M (=)
Key Generation Algorithm Identifier	M	M (=)		
Calling Certificate Path	M	M (=)		
Called Certificate Path			M	M (=)
Calling Key Generation Algorithm Parameters	U	C (=)		
Called Key Generation Algorithm Parameters			U	C (=)
Calling SAID	M	M (=)	M (=)	M (=)
Called SAID			M	M (=)
Result			M	M (=)

[Insert a discussion of each parameter. Note that the certificate path can be a single certificate.]

6.2.4. Send Key (SEND-KEY)

The SEND-KEY primitive forwards a symmetric cryptographic key to the remote key management end system. The cryptographic keying material is enciphered in a key encrypting key held by the remote KMAP, and the cryptographic keying material may have been obtained from a Center for Key Distribution (CKD) or a Center for Key Translation (CKT). SEND-KEY provides a confirmed service. If there is not an application association with the remote key management end system, then one is created.

The parameters of SEND-KEY are defined in the following table:

SEND-KEY Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)		
Called AE-Title	M	M (=)	M	M (=)
Encipherment Algorithm Identifier	M	M (=)		
Enciphered Cryptographic Keying Material	M	M (=)		
Calling SAID	M	M (=)	M (=)	M (=)
Called SAID			M	M (=)
Result			M	M (=)

The remote KMAP has previously obtained the key encrypting key that it uses to decrypt the Enciphered Cryptographic Keying Material. If this material came from a CKD, or a CKT, the key encrypting key also came from the respective center, through a manual distribution.

[Insert a discussion of each parameter.]

6.2.5. Negotiate Security Association Attributes (PICK-SA-ATTRS)

The PICK-SA-ATTRS primitive forms a security association by assigning attributes to a cryptographic key that was established using the SELECT-KEY, MAKE-KEY, or SEND-KEY primitive. The same application association used for the SELECT-KEY, MAKE-KEY, or SEND-KEY primitive will be used to provide this confirmed service. The enabling of cryptographic keying material and associated attributes within the SMIB for use by the Security Protocol is a local matter; however, when the PICK-SA-ATTRS.confirm is provided, the cryptographic keying material and associated attributes has been enabled for use by the Security Protocols on both of the peer key management end systems.

The PICK-SA-ATTRS exchange is protected by the previously established security association. This protection provides peer entity authentication; the protection also provides confidentiality, integrity, or confidentiality and integrity.

The parameters of PICK-SA-ATTRS are defined in the following table:

PICK-SA-ATTRS Parameter Name	Req	Ind	Rsp	Cnf
Enciphered Security Protocol Attributes List	M	M (=)		
Enciphered Security Protocol Attributes			M	M (=)
Calling SAID			M	M (=)
Called SAID	M	M (=)		
Result			M	M (=)

[Insert a discussion of each parameter.]

[Concerns have been raised about the use of the term Lower Layer. However, the scope of this document clearly states that the KMP entity supports Security Protocols. Also today, the only security protocols that exist that require key management support operate at the lower layers 4, 3, and 2.]

6.2.6. Spawn Key (SPAWN-KEY)

The SPAWN-KEY primitive forms a security association by transforming a previously established symmetric cryptographic key to form a new symmetric cryptographic key and assigning the same attributes with the new key as were assigned to the previously established key. Of course, the security association identifier attribute will be different for the two security associations. SPAWN-KEY provides a confirmed service. If there is not an application association with the remote key management end system, then one is created. The same symmetric cryptographic key is established in the SMIB of each key management end system. The enabling of cryptographic keying material and associated attributes within the SMIB for use by the Security Protocol is a local matter; however, when the SPAWN-KEY.confirm is provided, the cryptographic keying material and associated attributes has been enabled for use by the Security Protocols on both of the peer key management end systems.

The SPAWN-KEYexchange is protected by the previously established security association. This protection provides peer entity authentication; the protection also provides confidentiality, integrity, or confidentiality and integrity.

The parameters of SPAWN-KEY defined in the following table:

SPAWN-KEY Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)		
Called AE-Title	M	M (=)	M	M (=)
Key Transformation Algorithm Identifier	M	M (=)		
Previously Established Calling SAID	M	M (=)		
Previously Established Called SAID	M	M (=)		
Calling SAID	M	M (=)	M (=)	M (=)
Called SAID			M	M (=)
Result			M	M (=)

[Insert a discussion of each parameter. Is commit needed to implemet this? Discussion must be added to allow the transformation of a confidentiality key and an integrity key. This is part of the Key Transformation Algorithm Identifier list supplied in the SPAWN-SA primitive.]

6.2.7. Get Multicast Key (GET-MKEY)

The GET-MKEY primitive forms a security association by obtaining an enciphered symmetric cryptographic key from the remote key center that will be used by the Security Protocol to protect broadcast or multicast communications. Each security protocol entity instigates obtaining the broadcast or multicast key as its needs dictates. The key center does not push the broadcast or multicast key out to every member of the group.

The GET-MKEY primitive may only be used after a SELECT-KEY, MAKE-KEY, or SEND-KEY primitive. The same application association used for the SELECT-KEY, MAKE-KEY, or SEND-KEY primitive will be used to provide this confirmed service. The security association identifier and all other security association attributes are assigned by the remote key management application so that the same ones will be used by all Security Protocol entities who posses this multicast key. The enabling of cryptographic keying material and associated attributes within the SMIB for use by the Security Protocol is a local matter; however, when the GET-MKEY.confirm is provided, the cryptographic keying material and associated attributes has been enabled for use by the Security Protocols in the SMIB of the local peer key management end system.

The GET-MKEY exchange is protected by the previously established security association. This protection provides peer entity and confidentiality; it may also provide integrity. A security association which provides integrity but not confidentiality must not be used.

The parameters of GET-MKEY are defined in the following table:

GET-MKEY Parameter Name	Req	Ind	Rsp	Cnf
Multicast Address List	M	M (=)		
Multicast Security Association List			M	M (=)
Calling SAID			M	M (=)
Called SAID	M	M (=)		
Result			M	M (=)

[The returned Multicast Security Association List includes the enciphered multicast key, the SAID, and the security association parameters for each multicast address requested.]

[Insert a discussion of each parameter. Should this be a KCASO primitve in addition to a KPASO primitive? Make sure to describe the G-bit in the multicast SAID.]

6.2.8. Delete Key (DELETE-KEY)

The DELETE-KEY primitive notifies the remote key management end system that the cryptographic keying material and associated attributes which define a particular security association are being deleted from the local SMIB. In general, the remote key management end system will also delete the cryptographic keying material and associated attributes from the remote SMIB. DELETE-KEY provides an unconfirmed service. If there is not an application association with the remote key management end system, then one is created.

The DELETE-KEY exchange is protected by the previously established security association. This protection provides data origin authentication; the protection also provides confidentiality, integrity, or confidentiality and integrity.

The parameters of DELETE-KEY are defined in the following table:

DELETE-KEY Parameter Name	Req	Ind
Calling AE-Title	M	M (=)
Called AE-Title	M	M (=)
Calling SAID	M	M (=)
Called SAID	M	M (=)

[Insert a discussion of each parameter.]

6.2.9. Release Peer Association (RELEASE-P)

The RELEASE-P primitive releases the application association with the remote key management end system.

[Steal the parameter description from ACSE service definition.]

6.2.10. Abort Peer Association (ABORT-P)

The ABORT-P primitive aborts the application association with the remote key management end system.

[Steal the parameter description from ACSE service definition.]

6.3. Key Center Application Service Object (KCASO) Services

The KCASO provides four services: REQUEST-KEY, TRANSLATE-KEY, PUSH-KEY, RELEASE-C, and ABORT-C.

6.3.1. Request Key (REQUEST-KEY)

The REQUEST-KEY primitive obtains a symmetric cryptographic key from a Center for Key Distribution (CKD). Two copies of the symmetric cryptographic key are obtained, one is enciphered in a previously distributed key encipherment key (KEK), and the other is enciphered in a KEK that was previously distributed to the remote key management end system. If there is not an application association with the CKD, then one is created.

The parameters of REQUEST-KEY defined in the following table:

REQUEST-KEY Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)		
CKD AE-Title	M	M (=)	M	M (=)
Called AE-Title	M	M (=)	M (=)	M (=)
Key Material Algorithm Identifier	M	M (=)		
Enciphered Key Material for Calling Party			M	M (=)
Enciphered Key Material for Called Party			M	M (=)
Result			M	M (=)

[The listing of the three encrypted return values this way seems strange. They are usually returned as one big ciphertext block where the stuff for the called part is super encrypted.]

[Insert a discussion of each parameter.]

6.3.2. Translate Key (TRANSLATE-KEY)

The TRANSLATE-KEY primitive sends a symmetric cryptographic key that was enciphered using a key encipherment key (KEK) that is shared between the key local management end system and the Center for Key Translation (CKT) and obtains the same symmetric cryptographic key encipherd in a KEK that is shared between the remote key management end system and the CKT. If there is not an application association with the CKT, then one is created.

The parameters of TRANSLATE-KEY defined in the following table:

TRANSLATE-KEY Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)		
CKT AE-Title	M	M (=)	M	M (=)
Called AE-Title	M	M (=)	M (=)	M (=)
Key Material Algorithm Identifier	M	M (=)		
Enciphered Key Material for Calling Party	M	M (=)		
Enciphered Key Material for Called Party			M	M (=)
Result			M	M (=)

[Insert a discussion of each parameter.]

6.3.3. Send Unsolicited Key (PUSH-KEY)

The PUSH-KEY primitive obtains a symmetric cryptographic key from a Center for Key Distribution (CKD). The symmetric cryptographic key is sent by the CKD without receiving a request for the cryptographic key. Two copies of the symmetric cryptographic key are delivered. One is enciphered in a key encrypting key (KEK) that was previously distributed to the local key management end system. The other copy is enciphered in a KEK that was previously distributed to the remote key management end system. The local key management application process is expected to forward the symmetric cryptographic key to the remote key management end system using the KPASO SEND-KEY primitive. If there is not an application association with the CKD, then one is created.

The parameters of PUSH-KEY defined in the following table:

PUSH-KEY Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)	M	M (=)
CKD AE-Title	M	M (=)		
Called AE-Title	M	M (=)	M (=)	M (=)
Key Material Algorithm Identifier	M	M (=)		
Enciphered Key Material for Calling Party	M	M (=)		
Enciphered Key Material for Called Party	M	M (=)		
Result			M	M (=)

[This looks strange. The returned structure should match the REQUEST-KEY as possible. Can we delete it altogether?]
[Why is this a confirmed service????]
[Insert a discussion of each parameter.]

6.3.4. Release Center Association (RELEASE-C)

The Release primitive releases the application association with the CKD or CKT.

[Steal the parameter description from ACSE service definition.]

6.3.5. Abort Center Association (ABORT-C)

The Abort primitive aborts the application association with the CKD or CKT.

[Steal the parameter description from ACSE service definition.]

7. SECURITY EXCHANGES

This section defines the security exchanges that implement the key management protocol.

```
-- IMPORT PROTECTED, SECURITY EXCHANGE,SECURITY
TRANSFORMATION,  PROTECTION MAPPING, enciphered FROM GULS
IMPORT Certificationpath, AlgorithmIdentifier  FROM X.509
```

7.1. Key Management Application Entity (KMAE) Security Exchanges

The KMAE does not generate security exchanges directly, but issues service requests to the KPASO and KCASO.

7.2. Key Peer Application Service Object (KPASO) Security Exchanges

7.2.1. Negotiate Key Management Algorithm (PICK-KM-ALG) Security Exchange

--pick-km-alg

```
pick-km-alg          SECURITY-EXCHANGE  ::= {
                 SE-ITEMS           {offered-km-alg, accepted-km-alg}
                 IDENTIFIER         {ieee sils kmp id-pick-km-alg (1)          } }
offered-km-alg       SEC-EXCHG-ITEM           ::= {
                 ITEM TYPE  Pick-KM-Alg-rq
                 ERRORS     {alg-negotiation-failed}
                 ITEM ID    1  }
accepted-km-alg      SEC-EXCHG-ITEM           ::=  {
                 ITEM TYPE  Pick-KM-Alg-rp
                 ITEM ID    2  }
alg-negotiation-failed      SE-ERROR ::={
                 PARAMETER          Result
                 ERROR CODE         1  }
Pick-KM-Alg-rq              ::= SEQUENCE {
                            KMTechniqueList,
                            SecurityDomainList OPTIONAL  }
Pick-KM-Alg-rp              ::= SEQUENCE {
                            KMTechnique,
                            SecurityDomain OPTIONAL  }

KMTechniqueList             ::=  SEQUENCE OF KMTechnique
KMTechnique                 ::= SEQUENCE OF AlgorithmIdentifier
SecurityPolicyList          ::= SEQUENCE OF SecurityPolicy
SecurityPolicy              ::= OBJECT IDENTIFIER
Result                      ::= INTEGER {
                            success(0),
                            failed(1)  }
```

7.2.2. Select Key (SELECT-KEY) Security Exchange

```
-- select-key
select-key          SECURITY-EXCHANGE   ::=
{
            SE-ITEMS            {proposed-key, installed-key}
            IDENTIFIER          {ieee sils kmp id-select-key (2)     }
}
proposed-key                SEC-EXCHG-ITEM          ::=
{
            ITEM TYPE           Select-Key-rq
            ERRORS              {select-key-failed}
            ITEM ID             1
}
installed-key       SEC-EXCHG-ITEM          ::=
{
            ITEM TYPE           Installed-Key-rp
            ITEM ID             2
}
select-key-failed   SE-ERROR ::={
            PARAMETER           Result
            ERROR CODE          1  }

Select-Key-rq               ::= SEQUENCE  {
                            Cryptographic_Keying_Material_Identifier,
                            Calling_SAID   }

Install-Key-rp              ::= SEQUENCE  {
                            Calling_SAID,
                            Called_SAID,
Result   }
```

7.2.3. Make Key (MAKE-KEY) Security Exchange

```
--make-key

make-key            SECURITY-EXCHANGE   ::=
{
            SE-ITEMS            {initiator-make,  responder-make}
            IDENTIFIER          {ieee sils kmp id-make-key (3)     }
}
initiator-make      SEC-EXCHG-ITEM          ::=
{
            ITEM TYPE  Make-Key-rq
            ERRORS     {make-key-failed}
```

```
                ITEM ID       1
}
responder-make    SEC-EXCHG-ITEM           ::=
{
                ITEM TYPE  Make-Key-rp
                ITEM ID       2
}
make-key-failed    SE-ERROR ::=
{
                PARAMETER       Result
                ERROR CODE      1
}

Make-Key-rq                ::= SEQUENCE {
keyGenAlgorithmID          KEY-GEN-ALG.&key-Gen-Alg-Id ( {ValidAlgIds} ),
                           CallingCertPath,
                           KEY-GEN-ALG.&Rq-Parms
                             ( {ValidAlgIds} {@key-Gen-Alg-Id} ),
                           CallingSAID   }

Make-Key-rp                ::= SEQUENCE {
                           CalledCertPath,
                           KEY-GEN-ALG.&Rp-Parms
                             ( {ValidAlgIds} {@key-Gen-Alg-Id} ),
                           CallingSAID,
                           CalledSAID   }
CallingCertPath            ::= CertificationPath
CalledCertPath             ::= CertificationPath
CallingSAID                ::= OCTET STRING
CalledSAID                 ::= OCTET STRING
```

7.2.4. Send Key (SEND-KEY) Security Exchange

```
-- send-key
send-key              SECURITY-EXCHANGE   ::=
{
                SE-ITEMS            {proposed-key, installed-key}
                IDENTIFIER          {ieee sils kmp id-send-key (4)      }
}
proposed-key               SEC-EXCHG-ITEM           ::=
{
                ITEM TYPE           Send-Key-rq
                ERRORS              {select-key-failed}
                ITEM ID             1
}

--Install-Key-rp is defined as part of the select-key exchange
```

```
select-key-failed      SE-ERROR ::=
{
               PARAMETER       Result
               ERROR CODE      1
}

Send-Key-rq                    ::= SEQUENCE {
                               Encipherment_Algorithm_Identifier,
                               Enciphered_Cryptographic_Keying_Material,
                               Calling_SAID    }
Encipherment_AI                ::-= AlgorithmIdentifier
Enciphered_Cryptographic_Keying_Material Protected
```

7.2.5. Negotiate Security Association Attributes (PICK-SA-ATTRS) Security Exchange

--pick-sa-attrs

```
pick-sa-attrs          SECURITY-EXCHANGE   ::=
{
               SE-ITEMS        {offered-attrs, accepted-attrs,commit-attrs}
               IDENTIFIER      {ieee sils kmp id-pick-sa-attrs (5)         }
}
offered-attrs          SEC-EXCHG-ITEM          ::=
{
               ITEM TYPE       Pick-SA-Attrs-rq
               ERRORS          {attrs-negotiation-failed}
               ITEM ID         1
}
accepted-attrs         SEC-EXCHG-ITEM          ::=
{
               ITEM TYPE       Pick-SA-Attrs-rp
               ERRORS          {attrs-negotiation-failed}
               ITEM ID         2
}
commit-attrs           SEC-EXCHG-ITEM          ::=
{
               ITEM TYPE       Pick-SA-Attrs-commit
               ITEM ID         3
}
attrs-negotiation-failed     SE-ERROR ::=
{
               PARAMETER       Result
               ERROR CODE      1
```

```
}

Pick-SA-Attrs-rq    ::=  SEQUENCE
{
                CalledSAID,
                PROTECTED {SPAttrsList, enciphered}
}
Pick-SA-Attrs-rp    ::= SEQUENCE
{
                CallingSAID,
                PROTECTED {SPAttrs, enciphered}

Pick-SA-Attrs-commit    ::= SEQUENCE
{
                CalledSAID,
                PROTECTED {Commit, enciphered}
}
SPAttrsList         ::=     SEQUENCE OF SPAttrs
SPAttrs             ::=     SEQUENCE
{
                spAttrs     OBJECT IDENTIFER,
                attrs       ANY DEFINED BY spAttrs OPTIONAL
}
Commit                      ::=  SEQUENCE
{
                        Result,
                        Nonce
}
```

7.2.6. Spawn Key (SPAWN-KEY) Security Exchange

```
--spawn-key
spawn-key                       SECURITY-EXCHANGE   ::=
{
                SE-ITEMS        {initiator-spawn-key, responder-spawn-key}
                IDENTIFIER      {ieee sils kmp id-spawn-key (6)   }
}
initiator-spawn-key             SEC-EXCHG-ITEM          ::=
{
                ITEM TYPE       Spawn-Key-rq
                ERRORS          {spawn-key-failed}
                ITEM ID         1
}
responder-spawn-key             SEC-EXCHG-ITEM          ::=
{
                ITEM TYPE       Spawn-Key-rp
                ITEM ID         2
```

```
}
spawn-key-failed                SE-ERROR ::=
{
            PARAMETER       Result
            ERROR CODE      1
}

Spawn-Key-rq                ::= SEQUENCE
      {                     Previously_Established_Called_SAID
                            PROTECTED {
                            SEQUENCE {
                                  Key_Transformation_Algorithm_Identifier,
                                  Previously_Established_Calling_SAID,
                                  Calling_SAID },    enciphered }
      }

Spawn-Key-rp                ::= SEQUENCE {
                            Previously_Established_Calling_SAID,
                            PROTECTED {
                            SEQUENCE {
                                  Called_AE-Title,
                                  Calling_SAID,
                                  Called_SAID,
                                  Result }, enciphered } }

Key_Transformation_Algorithm_Identifier        ::= OBJECT IDENTIFIER
Previously_Established_Calling_SAID            ::= OCTET STRING
Previously_Established_Called_SAID             ::= OCTET STRING
Called_SAID                                    ::= OCTET STRING
Calling_SAID                                   ::= OCTET STRING
```

7.2.7. Get Multicast Key (GET-MKEY) Security Exchange

```
--get-multicast-key
get-multicast-key           SECURITY-EXCHANGE  ::=
{
            SE-ITEMS        {request-mcast-key, mcast-key-installed}
            IDENTIFIER      {ieee sils kmp id-get-mcast-key (7)       }
}
request-mcast-key           SEC-EXCHG-ITEM         ::=
{
            ITEM TYPE       Get-Mkey-rq
            ERRORS          {get-mcast-key-failed}
            ITEM ID         1
}
mcast-key-installed         SEC-EXCHG-ITEM         ::=
{
```

```
                ITEM TYPE       Get-Mkey-rp
                ITEM ID         2

get-mcast-key-failed    SE-ERROR ::=
{
                PARAMETER       Result
                ERROR CODE      1
}

Get-Mkey-rq             ::=  SEQUENCE {
                        Called_SAID
                        PROTECTED {Mcast_Address_List, enciphered}  }

Get-Mkey-rp             ::= SEQUENCE  {
                        Calling_SAID
                        PROTECTED {
                                SEQUENCE  {
                                Mcast_SA-List,
                                Result  },
                                        enciphered}  }
```

7.2.8. Delete Key (DELETE-KEY) Security Exchange

```
delete-key              SECURITY-EXCHANGE  ::=
{
                SE-ITEMS        {request-delete-key}
                IDENTIFIER      {ieee sils kmp id-delete-key (8)   }
}
request-delete-key      SEC-EXCHG-ITEM          ::=
{
                ITEM TYPE       Delete-Key-rq
                ERRORS          {delete-key-failed}
                ITEM ID         1
}
-- Since this is an unconfirmed service, why do we have an SE-ERROR
delete-key-failed       SE-ERROR ::=
{
                PARAMETER       Result
                ERROR CODE      1
}

Delete-Key-rq           ::= SEQUENCE  {
                        Called_SAID,
                        PROTECTED { Calling_SAID, enciphered }
```

7.3.1. Request Key (REQUEST-KEY) Security Exchange

```
request-key         SECURITY-EXCHANGE   ::=
{
                SE-ITEMS            {key-request, key-response}
                IDENTIFIER          {ieee sils kmp id-request-key (9) }
}
key-request         SEC-EXCHG-ITEM          ::=
{
                ITEM TYPE           Request-Key-rq
                ERRORS              {request-key-failed}
                ITEM ID             1
}
key-response        SEC-EXCHG-ITEM          ::=
{
                ITEM TYPE           Request-Key-rp
                ITEM ID             2


request-key-failed  SE-ERROR ::=
{
                PARAMETER           Result
                ERROR CODE          1
}

Request-Key-rq      ::= SEQUENCE {
                        Nonce,
                        Key_Material_Algorithm_Identifier   }

Request-Key-rp      ::= SEQUENCE {
                        PROTECTED {
                          Enciphered_Key_Material_for_Calling_Party,
                              enciphered },
                        PROTECTED {
                          Enciphered_Key_Material_for_Called_Party,
                              enciphered },
                        PROTECTED {
                          Enciphered_Nonce,
                              enciphered },
                        Result  }
```

7.3.2. Translate Key (TRANSLATE-KEY) Security Exchange

```
translate-key       SECURITY-EXCHANGE   ::=
{
                SE-ITEMS            {translate-request, translate-response}
                IDENTIFIER          {ieee sils kmp id-translate-key (10)       }
}
translate-request           SEC-EXCHG-ITEM          ::=
```

```
{
                ITEM TYPE           Translate-Key-rq
                ERRORS              {translate-key-failed}
                ITEM ID             1
}
translate-response          SEC-EXCHG-ITEM          ::=
{
                ITEM TYPE           Translate-Key-rp
                ITEM ID             2


translate-key-failed        SE-ERROR ::=
{
                PARAMETER           Result
                ERROR CODE          1
}

Translate-Key-rq    ::= SEQUENCE {
                            Key_Material_Algorithm_Identifier,
                            PROTECTED {
                              Enciphered_Key_Material_for_Calling_Party,
                                   enciphered }  }

Translate-Key-rp    ::= SEQUENCE {
                            PROTECTED {
                              Enciphered_Key_Material_for_Called_Party,
                                   enciphered },
                            Result  }
```

7.3.3. Send Unsolicited Key (PUSH-KEY) Security Exchange

```
push-key        SECURITY-EXCHANGE  ::=
{
                SE-ITEMS            {push-key-request, push-key-response}
                IDENTIFIER          {ieee sils kmp id-push-key (11)   }
}
push-key-request            SEC-EXCHG-ITEM          ::=
{
                ITEM TYPE           Push-Key-rq
                ERRORS              {push-key-failed}
                ITEM ID             1
}
push-key-response           SEC-EXCHG-ITEM          ::=
{
                ITEM TYPE           Push-Key-rp
                ITEM ID             2
```

```
push-key-failed          SE-ERROR ::=
{
            PARAMETER       Result
            ERROR CODE      1
}

Push-Key-rq              ::= SEQUENCE {
                         Key_Material_Algorithm_Identifier,
                         PROTECTED {
                           Enciphered_Key_Material_for_Calling_Party,
                               enciphered },
                         PROTECTED {
                           Enciphered_Key_Material_for_Called_Party,
                               enciphered }  }

Push-Key-rp              ::= SEQUENCE {
                         Result  }
```

8. KEY GENERATION ALGORITHM OBJECT CLASS

```
KEY-GEN-ALG ::= CLASS
{
                                &Rq-Parms,
                                &Rp-Parms,
  -- Parameters syntax is provided as part of registering the key-gen-alg
  &key-Gen-Alg-Id              Identifier    UNIQUE
}
WITH  SYNTAX
{
  RQ-PARMS                     &Rq-Parms
  RP-PARMS                     &Rp-Parms
  ALG-ID                       &key-Gen-Alg-Id
}

Identifier ::= OBJECT IDENTIFIER
```

9. SECURITY TRANSFORMATIONS

All of the security transformations used in this standard are defined in GULS.

APPENDIX A
LOCATING SDE KEY MANAGEMENT ENTITIES
PROBE

[Does this text belong here or in IEEE 802.10d? The processing described here is not part of the application layer protocol, but it is used to locate (find the address of) the key management entity for a particular MAC address. That sounds like part of key management...]

[This description assumes that key management communications pass through SDE. That is, key management communications require a "bypass" security association between each pair of KMAEs. Is this really needed?]

ARCHITECTURE

The processing of "probe" packets to support distributed key management can be modeled as a SDE Key Management Entity (SKME) as shown in Figure A1. All of the probe packets contain the SDE designator as well as the individual bootstrap SAID for key management. This allows SDE to transparently deliver any PDUs containing the key management bootstrap SAIDs to the Key Management
SAP.

TRANSMISSION PROCESSING

This section describes the protocol processing associated with the transmission of an SDE PDU across a MAC bridge protected by the SDE protocol. The scenario is shown in Figure 3. For this example, the PDU is transmitted from MAC address A (Party A) to MAC address C (Party C).

1) The PDU is transmitted across Red Network N1. The terminology RED means
that information passing across this network is unencrypted.

2) Upon arrival at Bridge X, the SDE determines that no security association exists that supports communication between A and C.

3) The event indicating no association from SDE is forwarded to Layer Management. Layer Management checks local security policy to see if any further action should be taken. If policy dicates that recovery should be attempted, then a service request (GET_REMOTE_KMAE) is sent to the Key Management short stack requesting the remote KMAE address.

4) The service request is received by the KMSS. The KMSS keeps a local

database that associates destination MAC addresses with the KMAEs that serve the addresses. If the remote address is found, it is returned in the GET_REMOTE_KMAE response. If it is not found, the PROBE.request shown in Figure A2 is generated by the KMSS.

5) The MAC C destination address is passed to SDE as the destination MAC address. This destination MAC address ensures that this packet will follow the same path as the data packet. As such, it should reach the SDE that protects MAC address C (SDE Y). The source MAC address is changed to indicate that the source is MAC X. The original SA is included in the packet. The application entity title is included in the packet to allow the remote KMSS (KMSS Y) to manage its database of KMAE addresses.

6) Upon reception of the PROBE.request packet, SDE Y should deliver it to its KMSS. The KMSS checks the SDE SMIB (Layer violation-->can we do this, or do we institute a new request to SDE?) to see if their is a security association between X and Y.

6a) Security Association between X and Y?

Yes: Return the KSAIDs for the association the PROBE.response with the Application address of KMAE Y.

No: If there isn't, then the KMSS checks to see if this association is being currently negotiated.

If currently negotiating,

Block and wait until the Key SAIDs (KSAIDs) have been established before returning the PROBE.response.

If not currently negotiating,

1) Generate a KSAID for Y. Create an association in the SDE SMIB that for the KSAID with no protection attributes.

2) Generate and send a PROBE.response with KMAE Y's SAID. The Application Entity Address of KMAE Y is also included in the response.

7) Upon reception of the PROBE.response, KMSS X checks the KSAID field. If the KSAID contains two KSAIDs, then it searches the SDE SMIB for the appropriate entry. If found, then the KMSS generates the GET_KMAE_ADDRESS.response with the MAC address pair and the Application Entity Address of KMAE Y. A PROBE.confirm is generated which echoes back the selected addresses.

If their is only a single KSAID, then the KMSS Y thinks that no association exists for the exchange. The KMSS X searches the SDE SMIB for an assoication between MAC X and MAC Y. If found, it is deleted. The KMSS X then generates a KSAID, creates a non-protected entry in its SDE SMIB, and then generates a PROBE.confirm containing the new KSAID for X. The KMAE X generates a GET_KMAE_ADDRESS.response which contains the Application Entity Address of KMAE Y and the MAC address pair.

Error Handling

This section describes the error conditions and when the PROBE.error should be generated.

The SKME resides at Layer 2 within the SDE sublayer. As such, it has total access to the SDE SMIB. Further, the SDE entity if enhanced to support distributed key management can request the services of the SKME. The SKME receives and transmits the probe packets described in this section. The purpose of the SKME is to ensure that the application layer KMAE has access to the application layer address of the KMAE that serves the SDE to which communication is requested.

As shown in Figure A1, the SKME has two primitives: GET_ADDRESS.request and GET_ADDRESS.response. Its interface to SDE uses the standard SDE_UNITDATA.request and SDE_UNITDATA.indication to pass probe packets.

GET_ADDRESS.requestSKMESKME.SKME ADDRESS SKMEhe packet to allow the remote SKME (SKME An Probe ID which should be a sequential counter is generated and placed in the PROBE.request packet. Its purpose is to associate the incoming PROBE.response packets with the appropriate request. Upon reception at the remote SKME, the concatenation of the MAC address of the originator (SKME X in our example) with the Probe ID should provide a unique identifier for the Probe exchanges. SDE Y should deliver it to its SKMESKME checks the SDE SMIB The purpose of this security association is to allow communication between KMAE X and KMAE Y to pass through SDE. In many cases, this association will specify no security processing by SDE since security for the KMAE will be provided at Layer 7. Some applications may require Layer 2 protection for the KMAE exchanges as well. If Layer 2 protection is needed, then the attributes of this association will specify the appropriate SDE security services. This KMAE to KMAE association is identified by a Security Association ID (SAID). This special purpose SAID is called a KMAE SAID (KSAID) in the following sections. in is not a security association, then the SKME K (The Probe.response is shown in Figure A3) The Probe ID in the Probe.request is copied into the Probe.response.SKMESKME The Probe.confirm is shown in Figure A4. only a single KSAID, then SKMEdid not find anKMAE to KMAE exchangeSKME SKME . The KMAE X generates a GET_

Figure A5 shows the format of the Probe.error packet. At present, the following error codes are defined:

1: Ghost Association. This error packet is returned instead of the Probe.confirm if the Probe.response indicated that an association existed for communication between the KMAEs, but the local SDE SMIB contains no such association. The KSAIDs in the Probe.response packet are copied into the parameter field. The originator of the PROBE.request has the option to repeat the PROBE.request transmission. Upon reception, of this PROBE.error packet, it is recommended that the receiver delete the association specified in the packet.

2: Invalid Probe ID. This error packet is generated in response to a PROBE.response if the Probe ID is not known or a PROBE.confirm if the concatenation of the Source MAC address and the Probe ID is not known. The transmitter of this error packet takes no further action. It is recommended that the receiver of this error abort the processing of the Probe sequence specified in the error message.

3: Protocol Type Error: This error packet is generated when a PROBE packet indicates a valid Probe ID; however the recipient expected a different Probe packet. For instance, if a Probe.request has been transmitted, it is expected that the next packet will either be a Probe.error or a Probe.response. If a Probe.confirm is received, then this error packet is generated. The receiver and transmitter of this error packet should abort the processing of the Probe sequence specified in the error message.

4: Unknown Probe Type: This error packet is generated when a PROBE packet indicates a Type field other than those contained in this section. The receiver and transmitter of this error packet should abort the processing of the Probe sequence specified in the error message.

APPENDIX B
CERTIFICATE REPLACEMENT

B.1 Service Definition

B.1.1 Replace-Certificate

Replace-Certificate offers the KPASO the means to obtain a replacement X.509 certificate and optionally the associated private material. Replace-Certificate is a confirmed service.

Replace-Certificate Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)		
Called AE-Title	M	M (=)	M (=)	M (=)
Certificate-To-Be-Replaced	M	M (=)		
Replacement-Material			M	M (=)
Result			M	M (=)

B.2 Object Class Definition

```
CERTIFICATE-REPLACEMENT          ::= CLASS
{
            &cr-Identifier                  Identitier UNIQUE,
                                            &Cr-Replacement-Material       }
```

B.3 Security Exchanges

```
certificate-replacement          SECURITY-EXCHANGE  ::=
{
            SE-ITEMS             {request-replacement, replacement-response}
            IDENTIFIER           {ieee sils kmp id-make-key (3)    }
}
request-replacement       SEC-EXCHG-ITEM          ::=
{
            ITEM TYPE  Replacement-rq
            ERRORS     {replacement-failed}
            ITEM ID    1
}
replacement-response      SEC-EXCHG-ITEM          ::=
{
            ITEM TYPE  Replacement-rp
            ITEM ID    2
}
replacement-failed        SE-ERROR ::=
```

```
{
            PARAMETER       Result
            ERROR CODE      1
}

Replacement-rq          ::= SEQUENCE {
                  Called_SAID,
                  PROTECTED {
replacement-id    CERTIFICATE-REPLACEMENT.&cr-identifier
                                    ({ValidAlgIds}),
                  CERTIFICATE-REPLACEMENT.&Cr-Replacement-Material
                              ({ValidAlgIds} {@replacement-id}),
                  enciphered }  }

Replacement-rp          ::= SEQUENCE {
                  Called_SAID,
                  PROTECTED {
replacement-id    CERTIFICATE-REPLACEMENT.&cr-identifier
                                    ({ValidAlgIds}),
                  CERTIFICATE-REPLACEMENT.&Cr-Replacement-Material
                              ({ValidAlgIds} {@replacement-id}),
                  enciphered }  }
```

APPENDIX C
COMPROMISED MATERIAL LISTS

C.1 Security Sevices

C.1.1 Request-CML

Request-CML offers the KPASO the means to obtain a Compromised Material List. Request-CML is a confirmed service.

Request-CML Parameter Name	Req	Ind	Rsp	Cnf
Calling AE-Title	M	M (=)		
Called AE-Title	M	M (=)	M (=)	M (=)
Compromised-Material-List			M	M (=)
Result			M	M (=)

C.2 Object Class Definition

C.3 Security Exchanges