IEEE Std 802.10c-1998
(Supplement to
IEEE Std 802.10-1992)

IEEE Standard

Interoperable LAN/MAN Security (SILS)

Key Management (Clause 3)

17 April 1998 SH94558

IEEE Std 802.10c-1998
(Supplement to IEEE Std 802.10-1992)

IEEE Standards for Local and Metropolitan Area Networks:

Supplement to Standard for Interoperable LAN/MAN Security (SILS)—

Key Management (Clause 3)

Sponsor

**LAN MAN Standards Committee
of the
IEEE Computer Society**

Approved 21 January 1998

IEEE Standards Board

Abstract: A cryptographic key management model and a key management OSI Basic Reference Model Application Layer protocol are specified.

Keywords: association control service element (ACSE), asymmetric cryptographic algorithm, center-based key distribution, certificate-based key distribution, cryptographic keying material, key management model, key management protocol (KMP), manual key distribution, symmetric cryptographic algorithm

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

ISBN 1-55937-955-3

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

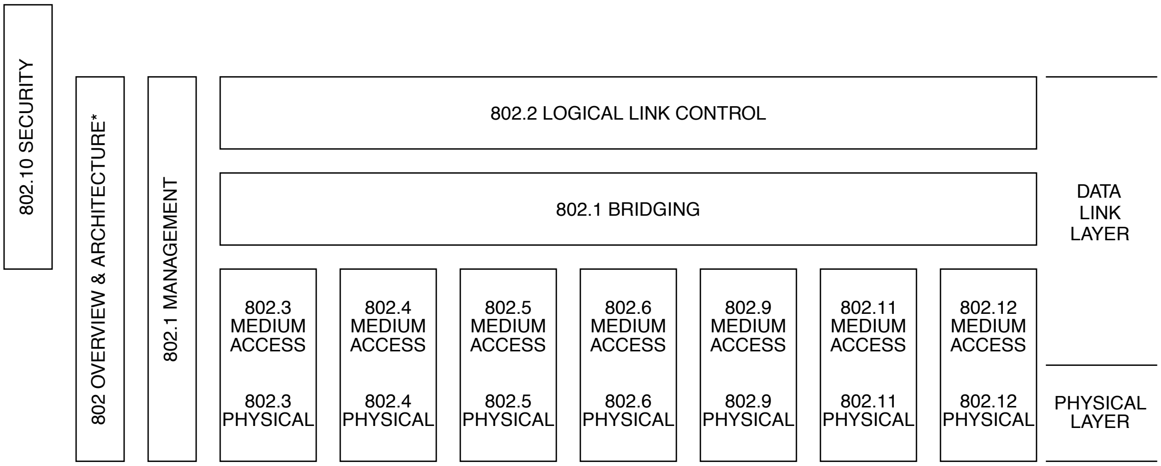
Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Introduction

[This introduction is not part of IEEE Std 802.10c-1998, IEEE Standards for Local and Metropolitan Area Networks: Supplement to Standard for Interoperable LAN/MAN Security (SILS)—Key Management (Clause 3).]

This standard is part of a family of standards for local and metropolitan area networks. The relationship between the standard and other members of the family is shown below. (The numbers in the figure refer to IEEE standard numbers.)



* Formerly IEEE Std 802.1A.

This family of standards deals with the Physical and Data Link layers as defined by the International Organization for Standardization (ISO) Open Systems Interconnection (OSI) Basic Reference Model (ISO/IEC 7498-1: 1994). The access standards define seven types of medium access technologies and associated physical media, each appropriate for particular applications or system objectives. Other types are under investigation.

The standards defining the access technologies are as follows:

- IEEE Std 802 — *Overview and Architecture*. This standard provides an overview to the family of IEEE 802 Standards.
- ANSI/IEEE Std 802.1B and 802.1k [ISO/IEC 15802-2] — *LAN/MAN Management*. Defines an OSI management-compatible architecture, and services and protocol elements for use in a LAN/MAN environment for performing remote management.
- ANSI/IEEE Std 802.1D [ISO/IEC 10038] — *Media Access Control (MAC) Bridges*. Specifies an architecture and protocol for the interconnection of IEEE 802 LANs below the MAC service boundary.
- ANSI/IEEE Std 802.1E [ISO/IEC 15802-4] — *System Load Protocol*. Specifies a set of services and protocol for those aspects of management concerned with the loading of systems on IEEE 802 LANs.
- ANSI/IEEE Std 802.1G [ISO/IEC 15802-5] — *Remote Media Access Control (MAC) Bridging*. Specifies extensions for the interconnection, using non-LAN communication technologies, of geographically separated IEEE 802 LANs below the level of the logical link control protocol.
- ANSI/IEEE Std 802.2 [ISO/IEC 8802-2] — *Logical Link Control*

- ANSI/IEEE Std 802.3 [ISO/IEC 8802-3] — *CSMA/CD Access Method and Physical Layer Specifications*
- ANSI/IEEE Std 802.4 [ISO/IEC 8802-4] — *Token Passing Bus Access Method and Physical Layer Specifications*
- ANSI/IEEE Std 802.5 [ISO/IEC 8802-5] — *Token Ring Access Method and Physical Layer Specifications*
- ANSI/IEEE Std 802.6 [ISO/IEC 8802-6] — *Distributed Queue Dual Bus Access Method and Physical Layer Specifications*
- ANSI/IEEE Std 802.9 [ISO/IEC 8802-9] — *Integrated Services (IS) LAN Interface at the Medium Access Control (MAC) and Physical (PHY) Layers*
- ANSI/IEEE Std 802.10 — *Interoperable LAN/MAN Security*
- IEEE Std 802.11 [ISO/IEC DIS 8802-11] — *Wireless LAN Medium Access Control (MAC) and Physical Layer Specifications*
- ANSI/IEEE Std 802.12 [ISO/IEC DIS 8802-12] — *Demand Priority Access Method, Physical Layer and Repeater Specifications*

In addition to the family of standards, the following is a recommended practice for a common Physical Layer technology:

- IEEE Std 802.7 — *IEEE Recommended Practice for Broadband Local Area Networks*

The following additional working group has authorized standards projects under development:

- IEEE 802.14 — *Standard Protocol for Cable-TV Based Broadband Communication Network*

Conformance test methodology

An additional standards series, identified by the number 1802, has been established to identify the conformance test methodology documents for the 802 family of standards. Thus the conformance test documents for 802.3 are numbered 1802.3.

IEEE Std 802.10c-1998

The IEEE 802.10 Working Group was formed in May 1988 to address the security of Local Area and Metropolitan Area Networks (LANs and MANs). Work on cryptographic key management began in May 1989. The IEEE 802.10 Working Group is sponsored by the LAN MAN Standards Committee. IEEE 802.10 provides interoperability standards that are compatible with existing IEEE 802 standards and Open Systems Interconnection (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 provide increased performance. As a result, there is a growing need to standardize data network protocols to ensure that data networks will interoperate effectively and securely.

As standards evolve, several key areas will become critically important. One of these areas is network security. 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 security standards also becomes a priority. Work on security standards has already started; and where applicable, this standard incorporates this previous work.

This standard contains state-of-the-art material. The area covered by this standard is undergoing evolution. Revisions are anticipated within the next few years to clarify existing material, to correct possible errors, and to incorporate new related material. Information on the current revision state of this and other IEEE 802 standards may be obtained from

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

IEEE 802 committee working documents are available from

IEEE Document Distribution Service
AlphaGraphics #35   Attn: P. Thrush
10201 N. 35th Avenue
Phoenix, AZ 85051
USA

Participants

The following is a list of participants in the Standard for Interoperable LAN/MAN Security effort of the IEEE Project 802 Working Group. Voting members at the time of publication are marked with an asterisk (*).

Kenneth G. Alonge,* *Chair*
Russell D. Housley,* *Vice Chair and Editor*
Joseph G. Maley,* *Recording Secretary*
Richard McAllister,* *Executive Secretary*

Asher Altman
Kurt Augustine
Kirk Barker
Kym D. Blair
James Coyle
Dane Elliot
Joanne Evans
Warwick Ford
Kim Kirkpatrick
Robert D. Kolacki
Paul Lambert
Wen-Pai Lu
Massimo Mascoli
Michael Michnikov
Ken McCoy
Tassos Nakassis
Noel A. Nazario
Barbara Patrick
Brian Phillips
James Randall
Karen Randall
Amy Reiss
Robert Rosenthal
Anthony Sacco
Brian P. Schanning
Emile W. Soueid
Jeffrey Vignes
Dale L. Walters
David Wheeler
Michael White
Joseph B. Williamson
Peter Yee
Roberto Zamparo
James E. Zmuda

The following persons were on the balloting committee:

William B. Adams
Kenneth G. Alonge
Alan Arndt
Kit Athul
Greg Bergren
James T. Carlo
Michael H. Coden
Robert S. Crowder
Peter Ecclesine
Gregory Elkmann
James T. Ellis
John E. Emrich
Philip H. Enslow
Changxin Fan
John W. Fendrich
Michael A. Fischer
Sandra J. Forney
Harvey A. Freeman
Robert J. Gagliano
Gautam Garai
Eli Herscovitz
Russell D. Housley
Henry Hoyt
Gilbert J. Huey
Thomas R. Hunwick
Richard J. Iliff
Cynthia E. Irvine
Peter M. Kelly
Gary C. Kessler
Yongbum Kim

Stephen Barton Kruger
Kenneth C. Kung
Lanse M. Leach
Randolph S. Little
Joseph C. J. Loo
Robert D. Love
Wen-Pai Lu
Joseph G. Maley
Richard K. McAllister
Darrell B. McIndoe
Milan Merhar
Steve Messenger
Bennett Meyer
Colin K. Mick
Ann Miller
Dale W. Miller
David S. Millman
Refik Molva
Warren Monroe
John E. Montague
Kinji Mori
James R. Moulton
Wayne D. Moyers
Michael John Nash
Noel Nazario
Dan Nessett
Paul Nikolich
Richard O'Brien
Robert O'Hara

Donal O'Mahony
Roger Pandanda
Vahid Parsi
Lucy W. Person
Andreas Pfitzmann
Thomas L. Phinney
Vikram Punj
Karen T. Randall
Edouard Y. Rocher
James W. Romlein
Floyd E. Ross
Christoph Ruland
Brian P. Schanning
Roger R. Schell
Norman Schneidewind
Donald A. Sheppard
Dan Shia
Alex Soceanu
Fred J. Strauss
Michael L. Sutherland
Efstathios D. Sykas
Geoffrey O. Thompson
Brian Tretick
Mark-Rene Uchida
Dale Walters
Frank J. Weisser
Clark Weissman
Raymond P. Wenig
Qian-Li Yang
Oren Yuen

The final conditions for approval of this supplement were met on 21 January 1998. This supplement was conditionally approved by the IEEE Standards Board on 9 December 1997, with the following membership:

Donald C. Loughry, *Chair*
Richard J. Holleman, *Vice Chair*
Andrew G. Salem, *Secretary*

Clyde R. Camp
Stephen L. Diamond
Harold E. Epstein
Donald C. Fleckenstein
Jay Forster*
Thomas F. Garrity
Donald N. Heirman
Jim Isaak
Ben C. Johnson

Lowell Johnson
Robert Kennelly
E. G. "Al" Kiener
Joseph L. Koepfinger*
Stephen R. Lambert
Lawrence V. McCall
L. Bruce McClung
Marco W. Migliaro

Louis-François Pau
Gerald H. Peterson
John W. Pope
Jose R. Ramos
Ronald H. Reimer
Ingo Rüsch
John S. Ryan
Chee Kiow Tan
Howard L. Wolfman

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
Alan H. Cookson

Valerie E. Zelenty
IEEE Standards Project Editor

Contents

IEEE Standards for Local and Metropolitan Area Networks:

Supplement to Standard for Interoperable LAN/MAN Security (SILS)—

Key Management (Clause 3)

Revisions to IEEE Std 802.10-1992

The contents of this document will be incorporated into IEEE Std 802.10 in a future edition. The clauses of this document are ordered to parallel the order of clauses in the base standard. This supplement is intended to be used in conjunction with IEEE Std 802.10-1992. Editing instructions necessary to incorporate this supplement into IEEE Std 802.10 are provided in ***bold italics.***

1.2.1 Acronyms

Add the following to 1.2.1 of IEEE Std 802.10-1992:

ACSE	Association Control Service Element
ASE	Application Service Element
ASO	Application Service Object
CA	Certification Authority
CCITT	International Telegraph Telephone Consultative Committee (Renamed ITU-T)
CF	Control Function
CML	Compromised Material List
CRL	Certificate Revocation List
GULS	Generic Upper Layers Security
ICV	Integrity Check Value
IEC	International Electrotechnical Commission

ISO	International Organization for Standardization
ITU-T	International Telecommunications Union - Technical
KCASO	Key Center Application Service Object
KDC	Key Distribution Center
KEK	Key Encryption Key
KMAE	Key Management Application Entity
KMAP	Key Management Application Process
KMID	Keying Material IDentifier
KMP	Key Management Protocol
KMTI	Key Management Technique Identifier
KPASO	Key Peer Application Service Object
KTC	Key Translation Center
MKC	Multicast Key Center
NLSP	Network Layer Security Protocol
PDU	Protocol Data Unit
SA	Security Association
SAID	Security Association IDentifier
SDE	Secure Data Exchange
SE	Security Exchange
SEI	Security Exchange Item
SESE	Security Exchange Service Element
SMIB	Security Management Information Base
ST	Security Transformation
TLSP	Transport Layer Security Protocol

1.3 References

Add the following references to IEEE Std 802.10-1992:

ANSI X9.30-1 (1995), Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry—Part 1: The Digital Signature Algorithm (DSA).[1]

ANSI X9.30-2 (1993), Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry—Part 2: The Secure Hash Algorithm (SHA).

ANSI DRAFT X9.42, Public Key Cryptography for the Financial Services Industry: Establishment of Symmetric Algorithm Keys Using Diffie-Hellman (draft dated 12 March 1998).

ANSI DRAFT X9.44, Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry: Transport of Symmetric Algorithm Keys Using RSA (draft dated September 1997).

ISO 7498-2: 1989, Information processing systems—Open Systems Interconnection—Basic Reference Model—Part 2: Security Architecture.[2] (ITU-T Recommendation X.800)

[1] ANSI publications are available from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

[2] ISO and ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse. ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

ISO 7498-3: 1997, Information technology—Open Systems Interconnection—Basic Reference Model—Part 3: Naming and Addressing.

ISO 8649: 1996, Information technology—Open Systems Interconnection—Service definition for the Association Control Service Element. (ITU-T Recommendation X.217)

ISO 8650-1: 1996, Information technology—Open Systems Interconnection—Connection-oriented protocol for the Association Control Service Element: Protocol specification.

ISO/IEC 7498-1: 1994, Information technology—Open Systems Interconnection—Basic Reference Model: The Basic Model. (ITU-T Recommendation X.200)

ISO/IEC 8824: 1990, Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1). (ITU-T Recommendation X.208)

ISO/IEC 9545: 1994, Information technology—Open Systems Interconnection—Application Layer structure. (ITU-T Recommendation X.207)

ISO/IEC 9594-8: 1995, Information technology—Open Systems Interconnection—The Directory—Part 8: Authentication framework. (ITU-T Recommendation X.509)

ISO/IEC 10181-1: 1996, Information technology—Open Systems Interconnection—Security frameworks for open systems: Overview. (ITU-T Recommendation X.810)

ISO/IEC 10736: 1995, Information technology—Telecommunication and information exchange between systems—Transport layer security protocol. (ITU-T Recommendation X.274)

ISO/IEC 10745: 1995, Information technology—Open Systems Interconnection—Upper layers security model. (ITU-T Recommendation X.803)

ISO/IEC 11577: 1995, Information technology—Open Systems Interconnection—Network layer security protocol. (ITU-T Recommendation X.273)

ISO/IEC 11586-1: 1996, Information technology—Open Systems Interconnection—Generic upper layers security: Overview, models and notation. (ITU-T Recommendation X.830)

ISO/IEC 11586-2: 1996, Information technology—Open Systems Interconnection—Generic upper layers security: Security Exchange Service Element (SESE) service definition. (ITU-T Recommendation X.831)

ISO/IEC 11586-3: 1996, Information technology—Open Systems Interconnection—Generic upper layers security: Security Exchange Service Element (SESE) protocol specification. (ITU-T Recommendation X.832)

ISO/IEC 11586-4: 1996, Information technology—Open Systems Interconnection—Generic upper layers security: Protecting transfer syntax specification. (ITU-T Recommendation X.833)

3. Key management

Replace Clause 3 and add the following subclauses to IEEE Std 802.10-1992:

3.1 Overview

3.1.1 Scope and purpose

This clause specifies a cryptographic key management model and protocol. The Key Management Protocol (KMP) is an OSI Basic Reference Model Application Layer protocol. KMP 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 KMP entity provides a service within the context of a defined security policy; it is outside the scope of this standard to define such security policies.

The KMP entity supports security protocol entities by managing the cryptographic keying material and security associations that are necessary for the provision of secure communication. KMP entities are responsible for creating the cooperative relationship and establishing the required cryptographic keying material. Security protocols supported by this KMP include the Secure Data Exchange security protocol, IEEE 802.10b; the Network Layer Security Protocol, ISO/IEC 11577; and the Transport Layer Security Protocol, ISO/IEC 10736. This KMP is intended to support other lower layer and upper layer security protocols, including those still under development by the Internet Engineering Task Force to provide security for the Internet Protocol (IP).

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

3.1.2 General

The KMP is used to create, spawn, and delete the security associations that are needed by security protocols as required by security policy. The security association is composed of symmetric keying material and security attributes.

Symmetric keying material is used in symmetric cryptographic algorithms. If two security protocol entities use the same symmetric algorithm, one security protocol entity uses the same cryptographic key to decrypt some or all of a protocol data unit (PDU) used by the other security protocol entity to encrypt the plaintext data.

The security attributes determine how the security protocol will use the symmetric keying material. For example, the security attributes tell which encryption algorithm and mode are used to provide confidentiality. Other security attributes tell which optional security protocol fields are present and which ones are absent.

This standard uses the functions and services defined in Generic Upper Layers Security (GULS), ISO/IEC 11586: 1996,[3] to provide an application-level communications infrastructure for the development of this mechanism-independent KMP.

GULS 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 SE is defined as a sequence of Security

[3] Information on references can be found in 1.3.

Exchange Items (SEIs). The Security Exchange Service Element (SESE), an application level element, uses ASN.1 notational tools to allow the consistent specification of SEs. GULS also defines security transformations (STs) to provide integrity, confidentiality, or both, of application data communicated between open systems. STs consist of a transfer-syntax function and ASN.1 notational tools. A transfer-syntax function, the Generic Protecting Transfer Syntax, allows applications to signal their integrity and confidentiality requirements in a communication protocol. ASN.1 notational tools 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 that satisfy these requirements.

This standard uses the functions defined in GULS to define SEs and STs that are applicable to the key distribution techniques discussed in subclause 3.2 (of this standard). 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.

3.1.3 Security definitions

3.1.3.1 keying material: Keys, codes, authentication material, or state information necessary to establish and maintain cryptographic relationships.

3.1.3.2 nonce: A unique identifier. Examples of a nonce are a random number, a timestamp, or a counter, each of which is used only once.

3.1.3.3 pairwise: Something shared only by two parties.

3.1.3.4 rekey: Replacement of keying material by manual or electronic means. In certificate-based key management, rekey is the issuing of a replacement certificate, and may include the replacement of the asymmetric key pair.

3.1.3.5 spawn: Given one security association, to create another uniquely named security association by changing the keying material, the security attributes, or both.

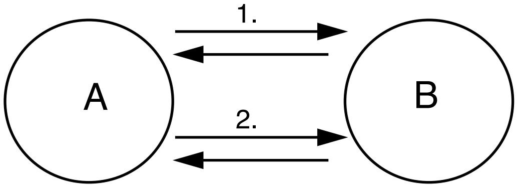
3.1.3.6 update: A one-way transformation of a symmetric cryptographic key to form another symmetric cryptographic key.

3.2 Key distribution techniques

This subclause describes the key distribution techniques that are supported by this protocol. Three classes of key distribution techniques are supported: manual distribution, center-based key distribution, and certificate-based distribution. The key center distribution techniques support both Key Distribution Centers (KDCs) and Key Translation Centers (KTCs). KDCs are sometimes called Centers for Key Distribution (CKDs); they are the same. KTCs are sometimes called Centers for Key Translation (CKTs); they are the same. The format of certificates used in the certificate-based distribution techniques is described in ITU-T Recommendation X.509.

3.2.1 Manual key distribution techniques

Manual key distribution techniques may be used to establish pairwise or multicast cryptographic keying material. For this technique, cryptographic keying material is generated a-priori and stored on the peer key management systems via means other than KMP. The cryptographic keying material is stored in a cache of memory and accessed using Keying Material Identifiers (KMIDs). See Figure 3-1. The cache is usually located in the Security Management Information Base (SMIB). The keying material cache with KMIDs is analogous to an array with indices. KMIDs are updated in the cache via means other than KMP.



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

Manual methods of key distribution can be cumbersome and have scalability issues. These methods use offline delivery to intended users of keying material that must be kept confidential. KMP allows the use of preplaced keying material or keying material generated by transforming preplaced keying material in the establishment and maintenance of security associations. The manual methods for preplacement of this keying material are beyond the scope of KMP.

In many cases, manual delivery of keying material is required only once per user. Distribution of additional keying material can be performed using the manually distributed key as a key encryption key (KEK) to encrypt the additional keying material. The encrypted keying material can then be distributed using any convenient method.

Manual key distribution methods do not provide any authentication other than that provided by the delivery method. Therefore, the strength of the procedures used for distribution of the keying material is extremely important.

Manual key distribution is suitable for multicast keying material distribution. In fact, it is often the most efficient way to distribute group or network-wide keying material, especially to large groups.

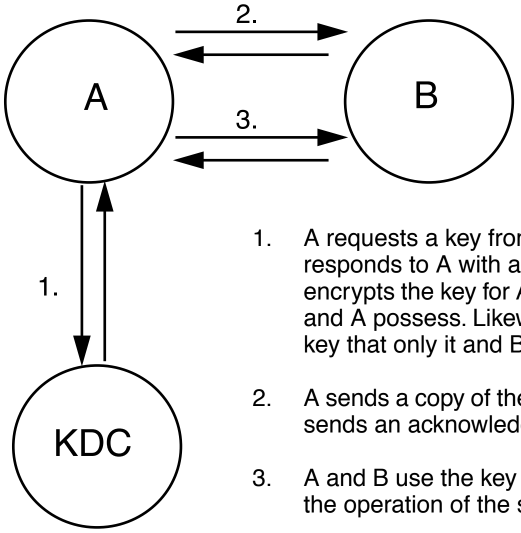
3.2.2 Center-based key distribution techniques

Center-based key distribution techniques may be used to establish pairwise or multicast cryptographic keying material between the participating parties via a trusted third party: the Key Distribution Center (KDC) or Key Translation Center (KTC).

Keying material established using a technique derived from the scheme originally described by Needham and Schroeder can be based on either a KDC or a KTC. The KDC and KTC protocols depend upon the manual distribution of KEKs to provide confidentiality and integrity protection for keying material. Dual-control or split-knowledge techniques may be used for initial secure distribution of KEKs, where two key components that will later be combined to form the cryptographic key are sent by separate paths. Each of the components conveys no knowledge of the final key. Once the manual KEKs have been distributed, cryptographic keying material may be electronically distributed. Confidentiality for the automatically distributed keys is provided by encryption using the manually distributed KEKs. Integrity may also be provided by the presence of an Integrity Check Value (ICV) or a digital signature.

Each party has a manually distributed secret KEK that is known only to itself and the KDC. One party, usually the initiator, requests the KDC to generate keying material to be shared by Party A and Party B. The

KDC generates the keying material and separately encrypts it in the manually distributed secret KEK for each party. The encrypted keying material is sent to the requesting party. The requesting party forwards the keying material to the other party encrypted in the other party's manually distributed secret KEK. As part of this transfer, the two parties demonstrate that they have received the same key from the KDC. See Figure 3-2.



1. A requests a key from the KDC to communicate with B. The KDC responds to A with a copy of the key for A and for B. The KDC encrypts the key for A with a key encryption key that only the KDC and A possess. Likewise, the KDC encrypts the key for B with a key that only it and B possess.
2. A sends a copy of the key received from the KDC to B, and then B sends an acknowledgment to A.
3. A and B use the key to protect the negotiation of the attributes for the operation of the security protocol.

Figure 3-2—Center-based key distribution

The keying material that is generated by the KDC can be used directly by a security protocol, or it can be used as a KEK. In the latter case, the KDC-generated KEK is used to protect keying material generated by one party while it is transferred to the other party. When this scheme is adopted, the three tiers of keying material are used as follows:

Tier 1: *Manually distributed KEK*. Pairwise between one party and the KDC.

Tier 2: *Automatically generated KEK*. Generated by the KDC and shared by Party A and Party B.

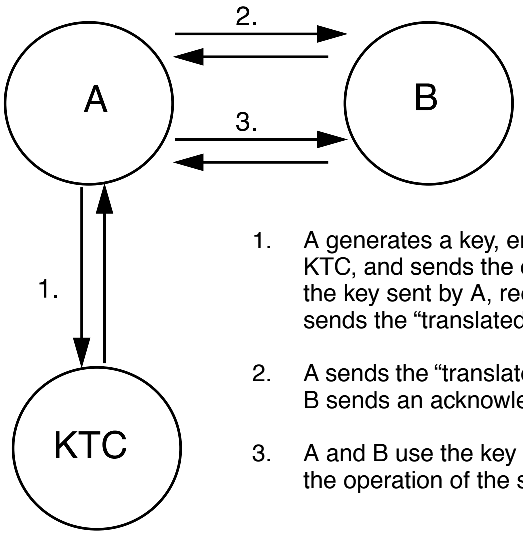
Tier 3: *Keying material used by the security protocol*. Generated by either Party A or Party B, and then sent to the other party encrypted in the automatically generated KEK (Tier 2).

Unlike KDCs, KTCs do not generate keying material. With KTCs, each party has a manually distributed secret KEK that is known only to itself and the KTC. Party A generates the keying material, encrypts it in the manually distributed secret KEK shared with the KTC, and sends the encrypted keying material to the KTC. The KTC decrypts the keying material, and then the KTC translates it by reencrypting it in the manually distributed secret KEK shared with Party B. The translated key is sent back to the requestor, Party A. Party A forwards the translated keying material to Party B. As part of this transfer, the two parties demonstrate that they have the same key. See Figure 3-3.

In cases where the communications topology prohibits Party A from communicating directly with the KDC or KTC, Party B can communicate with the KDC or KTC and send the encrypted keying material to Party A.

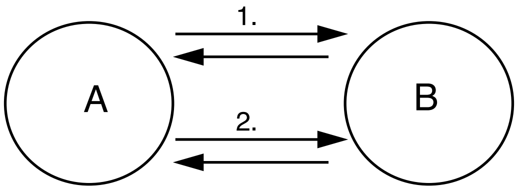
3.2.3 Certificate-based distribution techniques

Certificate-based key distribution techniques may be used to establish pairwise cryptographic keying material. This standard does not support the use of certificate-based key distribution techniques to establish multicast cryptographic keying material; however, once pairwise cryptographic keying material is established, it can be used to protect the distribution of multicast cryptographic keying material. See Figures 3-4 and 3-5.



1. A generates a key, encrypts the key in the KEK shared with the KTC, and sends the encrypted key to the KTC. The KTC decrypts the key sent by A, reencrypts it in the KEK shared with B, and sends the "translated" key to A.
2. A sends the "translated" key received from the KTC to B, and then B sends an acknowledgment to A.
3. A and B use the key to protect the negotiation of the attributes for the operation of the security protocol.

Figure 3-3—Center-based key translation



1. A and B exchange certificates, which they use to obtain 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 3-4—Certificate-based key distribution

There are two certificate-based distribution techniques:

- A public key cryptographic technique that uses asymmetric cryptography to encrypt locally generated keying material to protect it while it is being sent to the remote key management end system. This is commonly called *key transfer* or *key transportation*.
- A public key cryptographic technique that cooperatively generates common symmetric cryptographic keying material at both the local and remote key management end system. This is commonly called *key exchange;* it is also called *key agreement*.

This standard uses the formats for and management of certificates defined in the Directory Authentication Framework, ITU-T Recommendation X.509 (ISO/IEC 9594-8: 1995). The X.509 certificate contains 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 authority (CA). By signing the certificate, the CA is binding the public component of the asymmetric key pair and the identity. That is, the entity named by the distinguished name must hold the private component of the asymmetric key pair.

Each party must have a trusted public key. Generation and distribution of this trusted public key is beyond the scope of this standard. A party validates a certificate, in part, by validating that the current date is within the validity period of the certificate and by validating the certificate's signature using the public key of the CA. Likewise, the party validates the CA's certificate, which contains its public key. This process continues until the party validates a certificate using the trusted public key. This list of certificates composes a certification path.

CAs may revoke a certificate if the binding between the public component of the asymmetric key pair and the identity is no longer appropriate or if the private component of the asymmetric key pair is disclosed. As part of the certificate validation process, parties must ensure that a certificate has not been revoked by contacting the CA or by checking a recent Certificate Revocation List (CRL) issued and signed by the CA.

When certificates are cached, their validity should be periodically confirmed to ensure that they have not expired or been revoked. If after checking the validity of a certificate it is found to be revoked, then the certificate should be discarded from the cache and security associations that were established using the revoked certificate should be deleted.

3.2.4 Multicast key distribution techniques

Multicast key distribution relies on one of the key distribution techniques, manual, center-based, or certificate-based, to establish pairwise cryptographic keying material between the local key management end system and the Multicast Key Center (MKC). Using the pairwise cryptographic material, the local key management system requests the broadcast or multicast keying material and its associated attributes from the MKC. The MKC determines if the local key management system is authorized to receive the requested multicast keying material; if it is authorized, the MKC sends the multicast keying material and its associated attributes to the local key management system.

Each security protocol entity instigates obtaining the broadcast or multicast keying material as its needs dictate. The MKC does not push the broadcast or multicast keying material to every member of the group. Figure 3-5 depicts multicast key distribution using the three key distribution techniques.

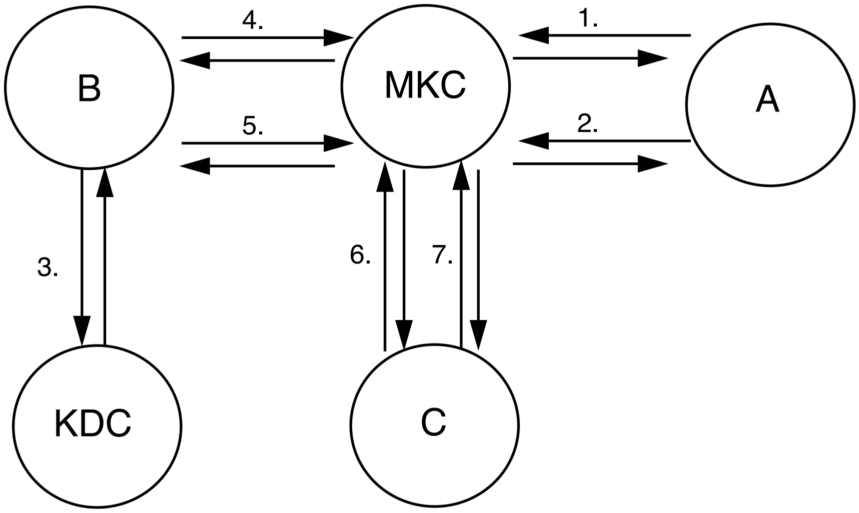
3.3 Key management model

The cryptographic keying material and security attributes that define a security association are represented as managed objects and stored in the SMIB. Security-related managed objects in the SMIB are available to security protocols across all layers of the OSI Reference Model. The KMP described here allows for the creating, spawning, and deleting of security associations within the SMIB. Creating a security association establishes new keying material and security attributes within the SMIB, spawning uses information from a security association in the SMIB to generate a new one, and deleting a security association makes the keying material and security attributes unavailable. Deleting a security association may remove the keying material or security attributes from the SMIB, if local security policy permits.

KMP supports two types of security associations: pairwise and multicast. Pairwise security associations support protected communication between two parties that share the same cryptographic keying material and security attributes. Multicast security associations support communication among two or more parties, where all parties share the same cryptographic keying material and security attributes.

3.3.1 Security association lifecycle

The creation of security associations is accomplished in two phases: establishment of cryptographic keying material, and negotiation of security attributes. Once cryptographic keying material is established, it is used to protect the security attribute negotiation.



1. A and the MKC select a key that was previously distributed manually.
2. A gets the multicast key from the MKC.
3. B requests a key from the KDC to communicate with the MKC.
4. B sends a copy of the key received from the KDC to the MKC. The key is protected with a key encryption key that only the MKC knows.
5. B gets the multicast key from the MKC.
6. C and the MKC exchange key generation information, which they use to make a secret key.
7. C gets the multicast key from the MKC.

Figure 3-5—Multicast key distribution

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. Spawn operations allow the generation of a new security association with new cryptographic keying material derived from that of the parent association and the same security attributes; the same cryptographic keying material and new security attributes; or new cryptographic keying material derived from that of the parent association and new security attributes. In every case, spawn operations generate a new security association with a new Security Association Identifier (SAID). Spawn operations do not affect the parent security association.

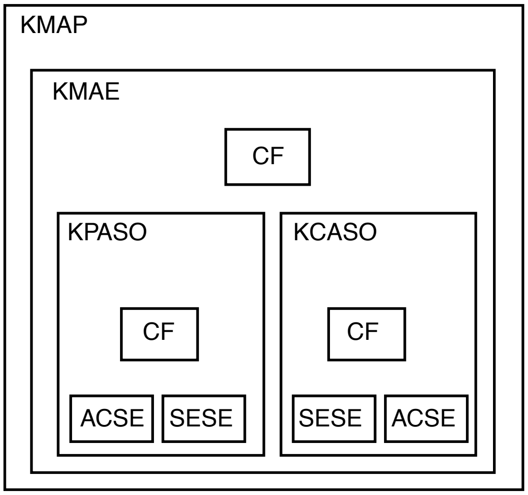
Either key management end system may determine that a security association is no longer needed. Once the determination is made, the local system notifies the remote system, and deletes the association. No confirmation is necessary.

3.3.2 Key management application entity structure

The key management application requires communication with remote key management applications. To support this communication, the KMP relies on OSI communication services provided by the ACSE and 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 negotiation technique is used with all three cryptographic keying material distribution techniques. This protocol is used to establish security associations that provide security services.

The Key Management Application Entity (KMAE) is an Application Service Object (ASO) that performs cryptographic keying material management. As shown in Figure 3-6, 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 KDC or a KTC to obtain cryptographic keying material. The Control Function (CF) coordinates interactions between the KPASO and the KCASO. The service interface to the KMAE is described in 3.4.1.



KMAP	Key Management Application Process
KMAE	Key Management Application Entity
KPASO	Key Peer Application Service Object
KCASO	Key Center Application Service Object
CF	Control Function
ACSE	Association Control Service Element
SESE	Security Exchange Service Element

Figure 3-6—Key management application structure

The KPASO comprises the ACSE, the SESE, and a CF. The ACSE is a common application service element defined by ISO/IEC 8649: 1996, and it establishes and terminates application-associations. The SESE is a common application service element defined by ISO/IEC 11586: 1996, and it performs security exchanges and security transformations. The SESE provides the SE-Transfer operation to the CF in order to transport a Security Exchange Item (SEI). Each protocol data unit in this protocol is represented as an SEI. The CF coordinates interactions between ACSE and SESE. The service interface to the KPASO is described in 3.4.2.

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 simplifies addressing.

The KCASO, like the KPASO, comprises the ACSE, the SESE, and a CF. Again, the CF coordinates interactions between ACSE and SESE. The service interface to the KCASO is described in 3.4.3.

The KMAE may invoke the KCASO multiple times, once for a KDC or for a KTC. Therefore, each KCASO invocation uses a single application-association. Again, limiting the KCASO to one application-association simplifies addressing.

3.3.3 Sequencing of application layer services

The KMAE does not implement any security exchanges; rather, the KMAE invokes the KPASO and the KCASO, which perform the necessary security exchanges. Communication between the KMAE and the KPASO, communication between the KMAE and the KCASO, and sequencing of these communications are controlled by the CF within the KMAE; however, these communications are an implementation detail and are not specified.

The KMAE may optionally use the KPASO to negotiate the key distribution technique and the algorithm that will be used to establish symmetric cryptographic keying material. Regardless of how the symmetric keying material is established, the KPASO uses the symmetric keying material to protect the security attribute negotiation exchange necessary 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 KDC, or to translate a key using a KTC. This symmetric cryptographic keying material is subsequently used by the KPASO to complete the creation of the security association.

3.3.3.1 Manually distributed key

To select cryptographic keying material from a collection of pre-positioned material or to select cryptographic keying material generated by transforming pre-positioned material, a Create-SA service request is issued to the KMAE.

In order to implement manually distributed cryptographic keying material, it is generated a-priori and stored on the peer key management systems via means other than KMP. The cryptographic keying material is stored in a memory cache and accessed using Keying Material Identifiers (KMIDs). Validity periods may be associated with the cached cryptographic keying material. The cache is usually located in the SMIB. The cache with KMIDs is analogous to an array with indices; see Table 3-1. If cryptographic keying material transformation is supported, then the cache must include a counter that tells the number of times a transformation has been applied to the manually distributed cryptographic keying material. Keeping this counter allows earlier versions of the cryptographic keying material, including the original, to be discarded when they are no longer needed.

Table 3-1—Cryptographic keying material cache

KMID	Cryptographic keying material	Transform counter
1	0x0123456789ABCDEF	0
2	0XF0DEBC9A78563412	48

The KPASO CF generates the proper SEIs from the 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. If an application-association is not already in place, the KPASO CF invokes the A-Associate service provided by ACSE. If the Create-SA parameters include a Key Management Technique Identifier (KMTI) List with multiple techniques (see 3.4.1.1.3), or the Security Policy Identifier must be announced (see 3.4.1.1.4), then the KMAE CF invokes the KPASO Pick-KM-Alg service. 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 the application-association in place; otherwise the KMAE CF invokes the Release-P service, which releases the application-association.

Figure 3-7 shows the relationship between the KMAE and the KPASO. As mentioned previously, 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.

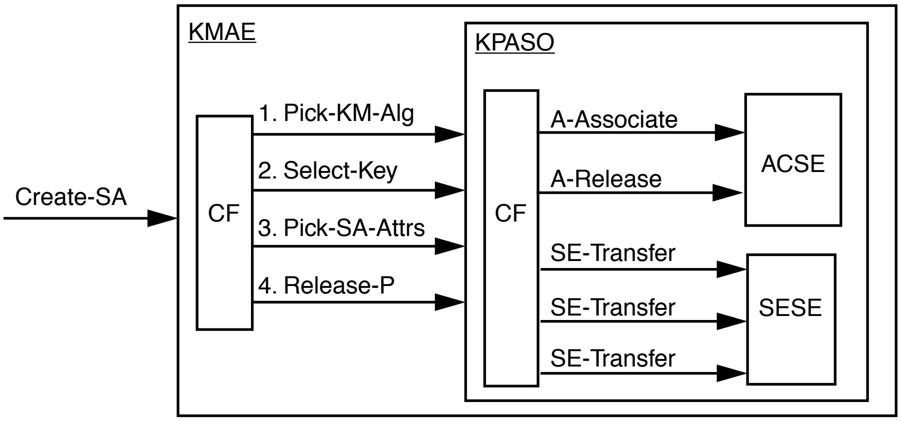


Figure 3-7—Create an SA by selecting manually distributed key

3.3.3.2 Key center distribution

A Create-SA service request is issued to the KMAE to obtain cryptographic keying material from a KDC or to translate cryptographic keying material using a KTC. If the Create-SA parameters include the KMTI List with multiple techniques (see 3.4.1.1.3), or the Security Policy Identifier must be announced (see 3.4.1.1.4), then the KMAE CF invokes the KPASO Pick-KM-Alg service. The KPASO generates the proper SEIs from the 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 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 or Translate-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 or translation center end system. If an application-association with the KDC or KTC 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 or Translate-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 invokes the Release-P and Release-C service, which releases the application-associations.

Figure 3-8 shows the relationship between the KMAE, the KPASO, and the KCASO. As mentioned previously, 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.

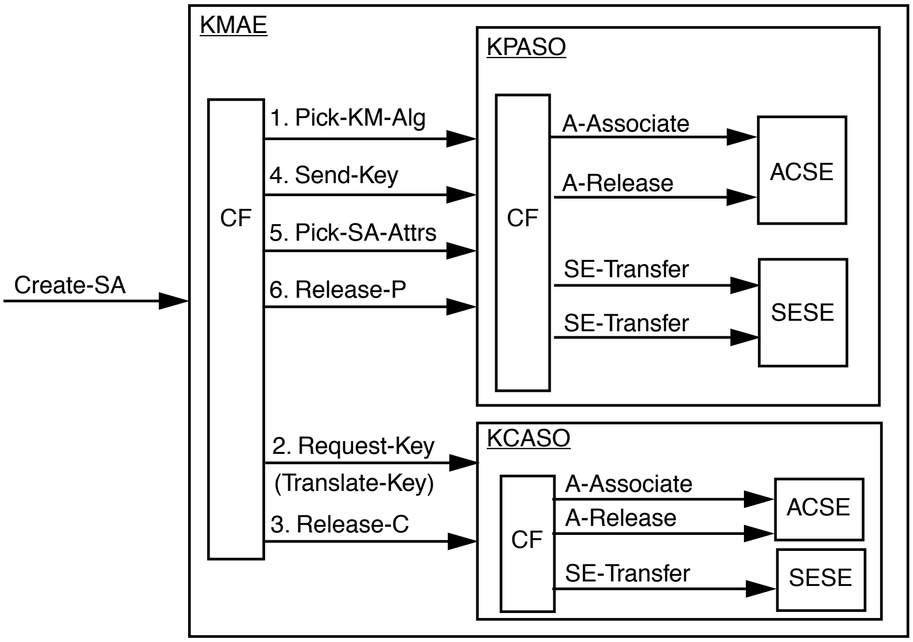


Figure 3-8—Create an SA using center-based key distribution

In cases where the communications topology prohibits the local key management end system from communicating directly with the KDC or KTC, the local KMAE CF issues the Please-Send-Key service to the KPASO requesting the remote key management end system to communicate with the KDC or KTC. In response, the remote KMAE CF issues the Request-Key or Translate-Key service to the remote KCASO, and then the remote KMAE CF issues the Send-Key service to the remote KPASO. Afterwards, the local 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 invokes the Release-P, which releases the application-association. If further key management communications are likely between the remote key management end system and the KDC or KTC, the remote KMAE CF leaves application-association in place; otherwise the remote KMAE CF invokes the Release-C, which releases the application-association.

3.3.3.3 Certificate-based key distribution

To generate a key using certificate-based techniques, a Create-SA service request is issued to the KMAE. If the Create-SA parameters include the KMTI List with multiple techniques (see 3.4.1.1.3) or the Security Policy Identifier must be announced (see 3.4.1.1.4), then the KMAE CF invokes the KPASO Pick-KM-Alg service. The KPASO generates the proper SEIs from the 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. 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 the application-association in place; otherwise the KMAE CF invokes the Release-P service, which releases the application-association.

Figure 3-9 shows the relationship between the KMAE and the KPASO. As mentioned previously, 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.

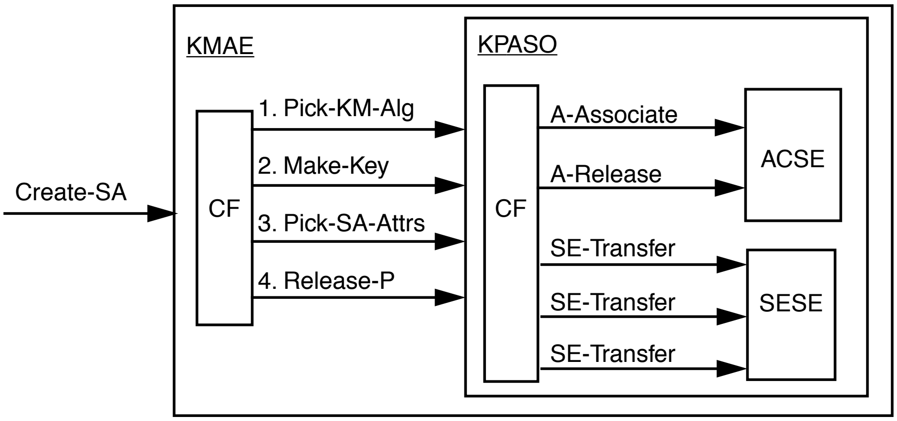


Figure 3-9—Create an SA using certificate-based key distribution

3.3.3.4 Multicast key distribution

The MKC generates the multicast keying material and the associated attributes, and then distributes them to the broadcast or multicast group members. The MKC determines the period of time that multicast keying material may be used. The KMAE uses the Create-MSA service to obtain multicast keying material from the MKC. The KMAE uses the Spawn-MSA service to obtain subsequent multicast keying material from the MKC.

There is no need for a Delete-MSA service. With pairwise security associations, the Delete-SA service is used to notify the remote KMAE that pairwise cryptographic keying material is being removed from the local SMIB. This notification is important in the pairwise case because communications associated with that cryptographic keying material are no longer possible. This is not the case with multicast keying material. When one KMAE deletes multicast keying material from the local SMIB, other members of the broadcast or multicast group can continue to communicate using that cryptographic keying material. The local deletion of multicast keying material has the effect of resigning from the broadcast or multicast group. The method used to delete multicast security associations is an implementation detail.

The security association identifier and all other security association attributes are assigned by the MKC so that the same ones will be used by all Security Protocol entities who possess the multicast keying material.

3.3.3.4.1 Create multicast security association

For Multicast keys, a Create-MSA service request is issued to the KMAE. If the Create-SA parameters include the KMTI List with multiple techniques (see 3.4.1.1.3), or the Security Policy Identifier must be announced (see 3.4.1.1.4), then the KMAE CF invokes the KPASO Pick-KM-Alg service. The KPASO generates the proper SEIs from the 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 the MKC. If an application-association is not already in place, the KPASO CF invokes the A-Associate service provided by ACSE.

Once the key distribution technique has been determined, the KMAE CF issues either the Make-Key, Select-Key, Send-Key, or Please-Send-Key service to the KPASO in order to establish keying material with the MKC. The Protected-Make-Key service may optionally be invoked following the completion of any of these services. After the KMAE CF has received confirmation from the KPASO for the chosen service, the KMAE CF issues the Get-MKey with a list of broadcast and multicast addresses. The MKC returns the list of approved broadcast and multicast addresses with a corresponding list of multicast tokens. The multicast token includes the multicast SAID. If further key management communications are likely, the application-association is left in place; otherwise the KMAE CF invokes the Release-P service, which releases the application-association.

Figure 3-10 shows the relationship between the KMAE and the KPASO. As mentioned previously, 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.

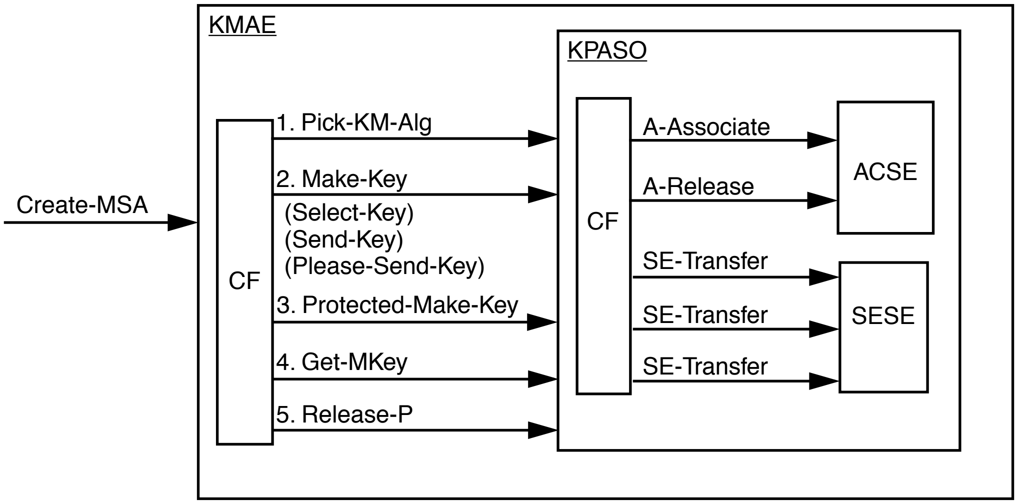


Figure 3-10—Multicast key distribution

3.3.3.4.2 Spawn multicast security association

To generate a new multicast security association from an existing one, a Spawn-MSA service request is issued to the KMAE. The KMAE CF uses a previously established multicast security association to protect the transfer of the new multicast token from the MKC. The KMAE CF invokes the KPASO service Get-Next-MKey to the MKC. The MKC returns the new multicast token, which includes the multicast SAID. The KPASO generates the proper SEIs from the passed parameters, and then passes the SEIs 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 the MKC. 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 the KMAE CF invokes the Release-P service, which releases the application-association.

Figure 3-11 shows the relationship between the KMAE and the KPASO. As mentioned previously, 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.

3.3.3.5 Spawn security association

A Spawn-SA service request is issued to the KMAE to generate a new security association from an existing security association by one of the following Spawn Options:

- Transform keying material and use the same attributes;
- Generate or transfer new keying material under the protection of an existing security association and use the same attributes;
- Transform keying material and negotiate new attributes;
- Generate or transfer new keying material under the protection of an existing security association and negotiate new attributes; or
- Use the same keying material, and negotiate new attributes.

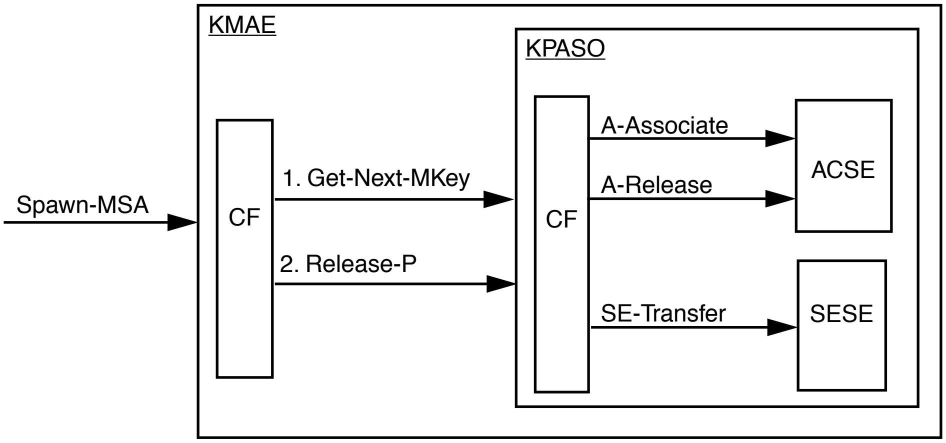


Figure 3-11—Multicast spawn

The KMAE CF uses information from the Spawn Option, the Key Transformation Algorithm Identifier, and the previously established calling and called SAIDs to invoke either the Spawn-Key, Protected-Make-Key, or Send-Key KPASO service. All of theses services assign a new SAID.

To change only the attributes while preserving the keying material, the KMAE must invoke the Spawn-Key service with an identity key transformation algorithm.

The KPASO generates the proper SEIs from the parameters, and then passes the SEIs 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 the KMAE CF invokes the Release-P service, which releases it.

Figure 3-12 shows the relationship between the KMAE and the KPASO. As mentioned previously, 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.

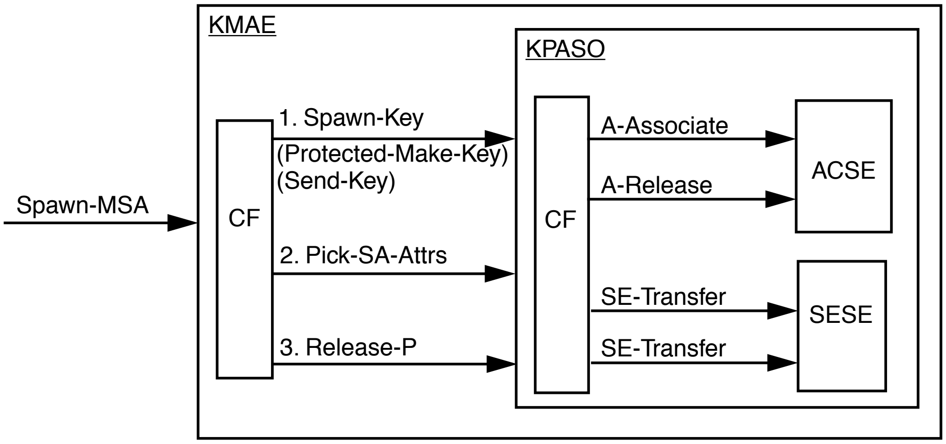


Figure 3-12—Spawn SA

3.3.3.6 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 parameters, and then passes the SEIs 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 the KMAE CF invokes the Release-P service, which releases it.

Figure 3-13 shows the relationship between the KMAE and the KPASO. As mentioned previously, 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.

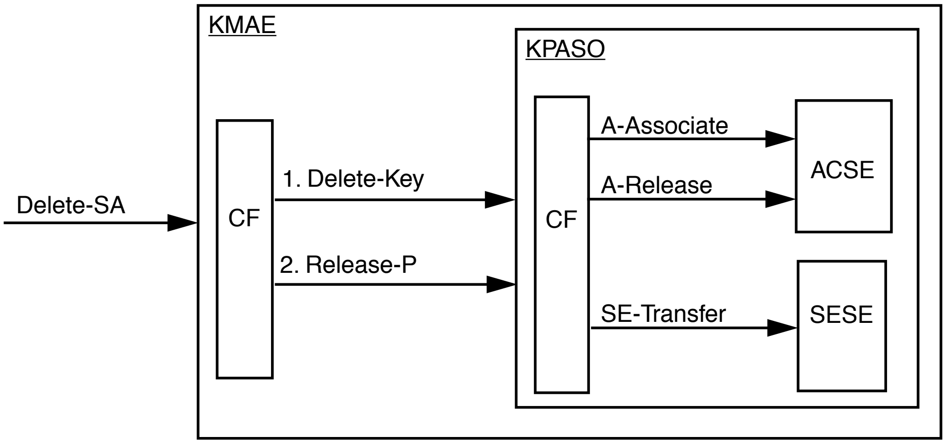


Figure 3-13—Delete SA

3.4 Service definition

The following abbreviations are used in this subclause:

M	Mandatory
U	User Optional
C	Conditional
(=)	If present, the parameter value must be the same as the parameter value in the previous column

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

3.4.1 Key management application entity (KMAE) services

The KMAE provides five services: Create-SA, Spawn-SA, Delete-SA, Create-MSA, and Spawn-MSA.

3.4.1.1 Create security association (Create-SA)

The Create-SA service 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 keying material depend on the Security Protocol that will use the security association. Issuing a Create-SA.conf announces that the security association is enabled on both end systems for use by the security protocol.

The parameters of Create-SA are defined in the Table 3-2.

Table 3-2—Create-SA parameters

Create-SA parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M	M (=)	3.4.1.1.2
Key Management Technique Identifier List	U	C (=)	—	—	3.4.1.1.3
Security Policy Identifier	U	C (=)	—	—	3.4.1.1.4
Security Association Attributes List	M	M (=)	—	—	3.4.1.1.5
Security Association Attributes	—	—	M	M (=)	3.4.1.1.6
Calling SAID	—	M	M (=)	M (=)	3.4.1.1.7
Called SAID	—	M	M (=)	M (=)	3.4.1.1.8
Result	—	—	M	M (=)	3.4.1.1.9

3.4.1.1.1 Calling AE-Title

The Calling AE-Title comprises the Application Title and Application Entity Qualifier of the calling KMAE, both of which are necessary during the application-association establishment and are passed from the calling key management application. The form of the AE-Title could either be the distinguished name of the application or an object identifier for the application.

```
CallingAETitle ::= AETitle

AETitle ::= CHOICE {
      AETitle-form1,
      AETitle-form2 }

AETitle-form1 ::= [0] SEQUENCE {
      COMPONENTS OF {
      APTitle-form1,
      AEQualifier-form1 }}

APTitle-form1 ::= IMPLICIT SEQUENCE OF
      RelativeDistinguishedName

AEQualifier-form1 ::= [1] IMPLICIT
      RelativeDistinguishedName

AETitle-form2 ::= [1] SEQUENCE {
      APTitle-form2,
      AEQualifier-form2 }

APTitle-form2 ::= [2] IMPLICIT OBJECT IDENTIFIER

AEQualifier-form2 ::= [3] IMPLICIT INTEGER
```

3.4.1.1.2 Called AE-Title

The Called AE-Title comprises the Application Title and Application Entity Qualifier of the called KMAE, both of which are necessary during the application-association establishment and passed from the calling key management application to indicate the intended called KMAE. The form of the AE-Title can either be the distinguished name of the application or an object identifier for the application.

```
CalledAETitle ::= AETitle -- AETitle is defined in 3.4.1.1.1
```

3.4.1.1.3 Key management technique identifier list

The Key Management Technique Identifier (KMTI) List contains a list of alternatives, in order of preference, regarding the type of key management distribution and the algorithms and values needed for key generation, confidentiality, integrity, and key transformation.

If the KMTI List contains more than one alternative or the Security Policy Identifier must be announced, then the KMAE CF must invoke Pick-KM-Alg. In many cases, the calling key management end systems support only a single key management technique; thus negotiation is unnecessary, and the KMAE CF may omit invocation of Pick-KM-Alg.

The KMTI is the selection made by the called KMAE from the list of alternatives in the KMTI List. The selection is the first acceptable alternative from the list in the KMTI List. The selection includes the type of key management distribution and the algorithms and values needed, if any, for key generation. Based on the value of the technique, the KMAE CF subsequently invokes Select-Key if the technique is equal to 0, Send-Key if the technique is equal to 1 (after it invokes either a Request-Key or a Translate-Key), Please-Send-Key if the technique is equal to 2, or Make-Key if the technique is equal to 3. Also, included in the selection is the confidentiality and integrity algorithms that the KMAE uses in subsequent exchanges during the establishment of the Security Association, i.e., within the Pick-SA-Attrs exchange. Also, the KMAE may subsequently use the key transformation algorithm within the Spawn-SA exchanges.

```
KMTechniqueIdList ::= SEQUENCE OF KMTechniqueId

KMTechniqueId ::= SEQUENCE  {
      technique          Technique,
      keyCreationAlg     KeyCreationAlg,
      attrConfidAlg      AttrConfidAlg,
      attrIntegrityAlg   AttrIntegrityAlg,
      transformationAlg  TransformationAlg OPTIONAL }

Technique  ::= INTEGER  {
      manual (0),
      requesterCallsCenter (1),
      responderCallsCenter (2),
      certificate (3)  }

KeyCreationAlg::= AlgorithmIdentifier
AttrConfidAlg::= AlgorithmIdentifier
```

```
AttrIntegrityAlg::= AlgorithmIdentifier
TransformationAlg::= AlgorithmIdentifier
```

3.4.1.1.4 Security policy identifier

The Security Policy Identifier is used by the calling KMAE to indicate a specific security policy to the called KMAE. In addition, authentication information can follow the policy identifier. The Security Policy Identifier is maintained in the SMIB, and it can be used to implicitly state security association requirements such as security service requirements, keying material selection, security protocol selection, and security protocol attributes. This parameter is optional.

```
SecurityPolicy ::= SEQUENCE  {
      policyId                OBJECT IDENTIFIER,
      policyVersion           INTEGER OPTIONAL,
      authenticationInfo      OCTET STRING OPTIONAL  }
```

3.4.1.1.5 Security association attributes list

The Security Association Attributes List contains information regarding the suite of security attributes required for the security association. For example, the security attributes could be the attributes for a security protocol. The information is placed in the list in order of preference by the calling KMAE. The values are determined by the security policy and maintained in the SMIB.

```
SAAttrsList ::= SEQUENCE of SAAttrs
```

3.4.1.1.6 Security association attributes

The Security Association Attributes is the selection made by the called KMAE from the Security Association Attributes List. The selection is made by the first match between the calling and called KMAE. The security attributes may be enumerated or they may be specified by referencing a previously established security association. If desired, padding may be included to make all of the alternatives approximately the same size. Any padding is ignored by the remote key management end system. An example of attributes for the SDE security protocol is contained in Annex 3E. The definition of the Object Class SP-ATTRS is contained in 3.5.5.2.

```
SAAttrs ::= CHOICE  {
      sameAs      [0]   ReferenceSA,
                  [1]   SPAttrs  }
ReferenceSA  ::= SEQUENCE  {
      callingSAID       Said,
      calledSAID        Said,
      padding           OCTET STRING OPTIONAL  }
SPAttrs  ::= SEQUENCE  {
      spAttrs           SP-ATTRS.&attrs-Id
      attrs             SP-ATTRS.&Sp-Attrs {@spAttrs} OPTIONAL  }
```

3.4.1.1.7 Calling SAID

The Calling SAID is a unique identifier for the Security Association and is determined by the calling KMAE. The calling and called KMAE determine their own SAIDs; therefore, the calling and called SAIDs may differ. The KMAE creates the SAID during the key establishment process.

The SAID is used by this KMP, and the format of the calling and called SAID is identical to the format of the SAID defined in 2.5.2.1.2 of IEEE Std 802.10-1992. If a security protocol requires its own SAID with a different format, then this SAID must be established with the selection of security attributes.

The SDE SAID format provides an indicator, the G-bit, for identifying security association types. For the security association created by this service, the KMAE must set the G-bit to FALSE, indicating a pairwise security association.

```
CallingSAID ::= OCTET STRING
```

3.4.1.1.8 Called SAID

The Called SAID is a unique identifier for the Security Association and is determined by the called KMAE. The calling and called KMAE determine their own SAIDs; therefore, the calling and called SAIDs may differ. The SAID is created during the key generation process.

```
CalledSAID :: = OCTET STRING
```

3.4.1.1.9 Result

The Result contains the exit status of the KMAE, KPASO, or KCASO services. A zero indicates success and one of the positive integers listed below indicates failure and corresponds to an error code.

```
Result ::= INTEGER {
success(0),
-- The service has completed successfully.

-- General Error Codes
failed_no_reason(1),
-- The service did not complete successfully with no reason given.
service_not_available(2),
-- The underlying service provider is not available.
service_not_available_at_this_time(3),
-- The underlying service is not available at the particular moment and
-- the caller is encouraged to try again at a later time.
illegal_ASN1_value_bad_value(4),
-- The ASN values passed from the calling party contain a bad value and
-- the call is rejected.
illegal_ASN1_value_missing_field(5),
-- The ASN values passed from the calling party do not contain fields
-- that are mandatory and the call is rejected.
illegal_ASN1_value_unknown_field(6),
-- The valid ASN values passed from the calling party contain an unknown
-- field and the call is rejected.
provider_abort_received(7),
-- The underlying service provider aborted causing the service to fail.
user_abort_received(8),
-- The user service on the local side initiated an abort causing the
-- service to fail.

-- KMP Specific Error Codes
negotiation_failure(64),
 -- The optional negotiation phase (Pick-KM-Algs) was not successful.
error_generating_key_material(65),
-- The key generation process (Select-Key, Send-Key, Make-Key, or
-- Protected-Make-Key) could not successfully generate the new keying
-- material.
error_retrieving_key_material(67),
-- The key retrieval process (Request-Key or Translate-Key) could not
-- successfully retrieve the new keying material from the key center.
not_ready_for_protected_service(68),
-- Even though the key generation process completed successfully, the
-- protective security mechanism was not ready.
error_retrieving_multicast_key(69),
-- The key retrieval process (Get-MKey or Get-Next-Mkey) could not
-- successfully retrieve the new multicast key from the MKC.
unknown_security_transformation(70),
-- The security transformation was unknown during the Spawn-Key service.
error_processing_SA_attributes(71),
-- The attribute negotiation process (Pick-SA-Attrs) was not successful.
invalid_called_SAID(72),
-- The called SAID was invalid.
invalid_calling_SAID(73),
-- The calling SAID was invalid.
release_peer_urgent(74),
-- The KPASO generated an urgent release request reason.
release_peer_user_defined_rq(75),
-- The KPASO generated an user defined release request reason.
release_peer_not_finished(76),
-- The KPASO generated a not finished release response reason.
release_peer_user_defined_rp(77),
-- The KPASO generated an user defined release response.
release_center_urgent(78),
-- The KCASO generated an urgent release request reason.
release_center_user_defined_rq(79),
-- The KCASO generated a user defined release request reason.
release_center_not_finished(80),
-- The KCASO generated a not finished release response reason.
release_center_user_defined_rp(81),
-- The KCASO generated a user defined release response.
abort_peer_user(82),
-- The KPASO generated an user abort.
abort_peer_provider(83),
-- The KPASO generated a provider abort.
abort_center_user(84),
-- The KCASO generated an user abort.
abort_center_provider(85),
-- The KCASO generated a provider abort.
expired_key(86),
-- The Select-Key request referenced keying material that has expired.
unavailable_key(87),
-- The referenced keying material is not available at this time.
transform_count(256-511)
-- Contains the number of transformations to the base keying material
-- needed to derive shared keying material. To determine this value,
-- subtract 256 from the error code received.
      }
```

3.4.1.2 Spawn security association (Spawn-SA)

The Spawn-SA service uses a previously established security association to establish a new security association. Spawn-SA can create a new security association from an existing security association in one of the following Spawn Options:

- Transform keying material and use the same attributes;
- Generate or transfer new keying material under the protection of an existing security association and use the same attributes;
- Transform keying material and negotiate new attributes;
- Generate or transfer new keying material under the protection of an existing security association and negotiate new attributes; or
- Use the same keying material, and negotiate new attributes.

Even if the keying material and all of the attributes are the same, the security association identifier is different for the two security associations. The previously established security association remains unaffected by the Spawn-SA service. Spawn-SA provides a 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 Spawn-SA.conf 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 may provide peer entity authentication. Other protection depends on the services provided by the previously established security association; confidentiality, integrity, or both may be provided.

The parameters of Spawn-SA are defined in Table 3-3.

Table 3-3—Spawn-SA parameters

Spawn-SA parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M	M (=)	3.4.1.1.2
Spawn Option	M	M (=)	M	M (=)	3.4.1.2.1
Key Transformation Algorithm Identifier	U	C (=)	—	—	3.4.1.2.2
Security Association Attributes List	U	C (=)	—	—	3.4.1.1.5
Security Association Attributes	—	—	U	C (=)	3.4.1.1.6
Previously Established Calling SAID	M	M (=)			3.4.1.2.3
Previously Established Called SAID	M	M (=)			3.4.1.2.4
Calling SAID	—	M	M (=)	M (=)	3.4.1.1.7
Called SAID	—	—	M	M (=)	3.4.1.1.8
Result	—	—	M	M (=)	3.4.1.1.9

3.4.1.2.1 Spawn option

The Spawn Option indicates whether the Spawn-SA service is to negotiate new attributes for use with existing keying material associated with a previously established security association, transform the symmetric keying material, make new keying material while under the protection of an existing security association, or transfer newly generated keying material under the protection of a previously established security association. The Spawn Option values are samekey for new attributes with existing key, update for transforming symmetric keying material, create for generating new keying material under the protection of an existing security association, and transfer for sending keying material under the protection of a previously established security association. The value for Spawn Option is selected by the Key Management Application. When the value is samekey (0), the KMAE invokes the Spawn-Key service with the identity key transformation algorithm, in order to assign a new SAID. When the value is update (1), the KMAE invokes the Spawn-Key service with the transformation algorithm identifier. When the value is create (2), the KMAE invokes the Protected-Make-Key service. When the value is transfer (3), the KMAE invokes the Send-Key service.

```
SpawnOption ::= INTEGER {
      samekey     (0),
      update      (1),
      create      (2),
      transfer    (3) }
```

3.4.1.2.2 Key transformation algorithm identifier

The Key Transformation Algorithm Identifier specifies the one-way transformation to be used by both the calling and called KMAEs to transform existing keying material into the new keying material for the security association. The Key Transformation Algorithm Identifier is set by the calling KMAE and is maintained in the SMIB or passed from the Key Management Application. The Key Transformation Algorithm Identifier is composed of an OBJECT IDENTIFIER and an optional INTEGER parameter, called transformCount. The transformCount specifies the number of times that the one-way transformation is applied to the original keying material.

The Key Transformation Algorithm Identifier was negotiated when the KMAE created the previous security association. If multiple keys comprise the previously established security association, the identifier tells the KMAE how to transform each key.

```
KeyTransformationAlgorithmIdentifier ::= AlgorithmIdentifier
```

3.4.1.2.3 Previously established calling SAID

The Previously Established Calling SAID is the unique identifier for the calling Security Association that is to be used as a template for the spawned security association. The previously established security association remains unaffected during the spawn process. The Previously Established Calling SAID and the Previously Established Called SAID must reference the same security association. The value for the Previously Established Calling SAID is maintained in the SMIB and is set by the calling KMAE.

```
PreviouslyEstablishedCallingSAID ::= OCTET STRING
```

3.4.1.2.4 Previously established called SAID

The Previously Established Called SAID is the unique identifier for the called Security Association that is to be used as a template for the spawned security association. The previously established security association remains unaffected during the spawn process. The Previously Established Calling SAID and the Previously Established Called SAID must reference the same security association. The value for the Previously Established Called SAID is maintained in the SMIB and is provided by the calling KMAE.

```
PreviouslyEstablishedCalledSAID ::= OCTET STRING
```

3.4.1.3 Delete security association (Delete-SA)

The Delete-SA service notifies the remote key management end system that the cryptographic keying material and associated attributes, which define a particular security association, are no longer available. Deleting a security association may remove the keying material and associated attributes from the SMIB, if local policy permits. Delete-SA provides an unconfirmed service. The remote key management end system also makes unavailable the cryptographic keying material and associated attributes, which define the security association. Again if local policy permits, deleting the security association may remove the cryptographic keying material and associated attributes from the remote SMIB.

Delete-SA is protected by the security association that is to be deleted, providing data origin authentication. The protection may also provide confidentiality, integrity, or both.

The parameters of Delete-SA are defined in Table 3-4.

Table 3-4—Delete-SA parameters

Delete-SA parameter name	.req	.ind	Reference
Calling AE-Title	M	M (=)	3.4.1.1.1
Called AE-Title	M	M (=)	3.4.1.1.2
Calling SAID	M	M (=)	3.4.1.1.7
Called SAID	M	M (=)	3.4.1.1.8

3.4.1.4 Create multicast security association (Create-MSA)

The Create-MSA service provides the capability for a peer key management system to obtain multicast tokens from the MKC. Create-MSA is a confirmed service. The KMAE first establishes cryptographic keying material with the MKC. Once established, the KMAE requests the multicast tokens associated with a list of multicast addresses. The MKC decides which multicast addresses are available to the peer key management systems, and then sends back the authorized list of multicast tokens. The KMAE requests tokens from the MKC by invoking the Get-Mkey service.

The parameters of Create-MSA are defined in Table 3-5.

Table 3-5—Create-MSA parameters

Create-MSA parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
MKCTitle	M	M (=)	M	M (=)	3.4.1.4.1
Key Management Technique Identifier List	U	C (=)	—	—	3.4.1.1.3
McastAddressList	M	M (=)	M (=)	M (=)	3.4.1.4.2
McastTokenList	—	—	M	M (=)	3.4.1.4.3
Result	—	—	M	M (=)	3.4.1.1.9

3.4.1.4.1 MKC title

The MKCTitle is the application layer title of the Multicast Key Center.

```
MKCTitle ::= AETitle
```

3.4.1.4.2 Mcast address list

The Mcast Address List is the list of multicast addresses for which the KMAP is requesting tokens.

```
McastAddressList ::= SEQUENCE OF McastAddress

McastAddress ::= OCTET STRING
```

3.4.1.4.3 Mcast token list

The MKC returns the McastToken to the requesting KMAE. The McastToken contains the mcastSAID. The mcastSAID is used by this KMP, and is identical to the format of the SAID defined in 2.5.2.1.2 of IEEE Std 802.10-1992. If a security protocol requires its own SAID with a different format, then this SAID must be established with the selection of security attributes.

The SDE SAID format provides an indicator, the G-bit, for identifying security association types. For the security association created by this service, the KMAE must set the G-bit to TRUE. If the format of the SAID of the security protocol has a multicast indicator, this indicator must be set to multicast for this service.

```
McastTokenList ::=  SEQUENCE OF McastToken

McastToken  ::= SEQUENCE  {
      mKCId       MKCTitle,
      PROTECTED   {  SEQUENCE  {
            mcastKey          OCTET STRING,
            mcastAddress      OCTET STRING,
            mcastAttributes   SPAttrs,
            mcastSAID         OCTET STRING,
            validityPeriod    ValidityPeriod  }, enciphered  },
                                    optionally-signed  }

ValidityPeriod  ::= SEQUENCE {
      notBefore   GeneralizedTime,
      notAfter    GeneralizedTime  }
```

3.4.1.5 Spawn multicast security association (Spawn-MSA)

The Spawn-MSA service provides the capability for a peer key management system to update a single multicast token using the protection of a previously established multicast security association. Spawn-MSA is a confirmed service that determines the previously established multicast SAID and sends the request to MKC. The MKC responds with the updated multicast token, based on the multicast token indicated by the previously established SAID, and the new subsequent multicast SAID.

The parameters of Spawn-MSA are defined in Table 3-6.

3.4.1.5.1 McastSAID

The McastSAID identifies the security association for which the KMAE is requesting an updated McastToken. The SDE SAID format provides an indicator, the G-bit, for identifying security association types. For the security association created by this service, the KMAE must set the G-bit to TRUE. If the format of the SAID of the security protocol has a multicast indicator, this indicator must be set to multicast for this service.

```
McastSAID ::= OCTET STRING
```

Table 3-6—Spawn-MSA parameters

Spawn-MSA parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
MKCTitle	M	M (=)	M	M (=)	3.4.1.4.1
McastSAID	M	M (=)	M (=)	M (=)	3.4.1.5.1
McastAddress	M	M (=)	M (=)	M (=)	3.4.1.4.2
McastToken	—	—	M	M (=)	3.4.1.4.3
Result	—	—	M	M (=)	3.4.1.1.9

3.4.2 Key peer application service object (KPASO) services

The KPASO provides thirteen services: Pick-KM-Alg, Select-Key, Make-Key, Send-Key, Pick-SA-Attrs, Spawn-Key, Get-MKey, Delete-Key, Release-P, Abort-P, Protected-Make-Key, Get-Next-MKey, and Please-Send-Key.

3.4.2.1 Negotiate key management algorithm (Pick-KM-Alg)

The KMAE uses the Pick-KM-Alg service to choose the key management algorithm that it will use to establish the same cryptographic symmetric key in the two key management end systems.

The KMAP may supply the KMTI List as a parameter in the Create-SA service request. If the KMAP did not supply the KMTI List, then the KMAE obtains the KMTI List from the SMIB. In either case if the KMTI List contains more than one alternative, then the KMAE CF must invoke Pick-KM-Alg. Also, if the Security Policy Identifier must be announced, then the KMAE CF must invoke Pick-KM-Alg. In many cases, the calling key management end system supports only a single key management technique and the Security Policy Identifier does not need to be announced; thus negotiation is unnecessary, and the KMAE may omit invocation of Pick-KM-Alg.

Since the KMTI negotiation performed by Pick-KM-Alg is not integrity protected, the KMTI is repeated in Pick-SA-Attrs to detect malicious modification during Pick-KM-Alg.

If there is not an application-association with the remote key management end system, then the KMAE creates one.

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 Table 3-7.

3.4.2.2 Select Key (Select-Key)

The Select-Key service chooses a symmetric cryptographic key from a cache of manually distributed keys that are stored in the SMIB. The cryptographic key is selected by indexing the cache using the Keying Material Identifier (KMID). If there is not an application-association with the remote key management end system, then one is created.

The KPASO must supply a nonce in the SEI passed to the SESE. The nonce is a unique identifier, such as a random number, timestamp, or counter, that is used once. The nonce is used to demonstrate liveness and prevent replay. The nonce supplied by the requester KPASO is passed to the responder KPASO, and then the

Table 3-7—Pick-KM-alg parameters

Pick-KM-alg parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M	M (=)	3.4.1.1.2
Key Management Technique Identifier List	U	C (=)	—	—	3.4.1.1.3
Key Management Technique Identifier	—	—	U	C (=)	3.4.1.1.3
Security Policy Identifier	U	C (=)	C (=)	C (=)	3.4.1.1.4
Result	—	—	M	M (=)	3.4.1.1.9

responder KPASO increments the nonce by one and passes the new value back to the requester KPASO. Since this exchange is protected, return of the expected value assures the requester KPASO that the responder KPASO has access to the specified cryptographic keying material. The responder KPASO could not generate the response in advance without knowledge of the nonce value, so liveness is demonstrated. The use of a unique nonce value ensures that the response is not a replay of a previous exchange.

The parameters of Select-Key are defined in Table 3-8.

Table 3-8—Select-Key parameters

Select-Key parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M	M (=)	3.4.1.1.2
Keying Material Identifier	M	M (=)	—	—	3.4.2.2.1
Key Transform Algorithm Identifier	U	C (=)	—	—	3.4.1.2.2
Calling SAID	M	M (=)	M (=)	M (=)	3.4.1.1.7
Called SAID	—	—	M	M (=)	3.4.1.1.8
Result	—	—	M	M (=)	3.4.1.1.9

3.4.2.2.1 Keying Material Identifier

The Keying Material Identifier names a pairwise key from the cache. The Keying Material Identifier contains the unique identifier that indicates the selected symmetric cryptographic key that is to be used. The calling KMAE retrieves the Cryptographic Keying Material Identifier from the SMIB.

```
KeyingMaterialIdentifier     ::= OCTET STRING
```

3.4.2.3 Make Key (Make-Key)

The Make-Key service obtains the data needed to establish a symmetric cryptographic key using a public key cryptographic technique. The public key technique may be a key distribution or a key agreement technique. The public keys are bound to the distinguished name of the key management end system in X.509 certificates. The distinguished name uniquely identifies the owner of the private key that corresponds to the public key contained in the certificate. 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 Table 3-9.

Table 3-9—Make-Key parameters

Make-Key parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M	M (=)	3.4.1.1.2
Key Generation Algorithm Identifier	M	M (=)	—	—	3.4.2.3.1
Calling Certificate Path	M	M (=)	—	—	3.4.2.3.2
Called Certificate Path	—	—	M	M (=)	3.4.2.3.3
Calling Key Generation Algorithm Parameters	U	C (=)	—	—	3.4.2.3.4
Called Key Generation Algorithm Parameters	—	—	U	C (=)	3.4.2.3.5
Calling SAID	M	M (=)	M (=)	M (=)	3.4.1.1.7
Called SAID	—	—	M	M (=)	3.4.1.1.8
Calling Attribute Certification Path	U	C (=)	—	—	3.4.2.3.6
Called Attribute Certification Path	—	—	U	C (=)	3.4.2.3.7
Result	—	—	M	M (=)	3.4.1.1.9

3.4.2.3.1 Key generation algorithm identifier

The Key Generation Algorithm Identifier contains the identifier to the key generation algorithm. This Key Generation Algorithm Identifier is obtained from the SMIB and can be optionally negotiated during the Pick-KM-Alg service.

```
keyGenAlgorithmID KEY-GEN-ALG.&key-Gen-Alg-Id
```

3.4.2.3.2 Calling certificate path

The Calling Certificate Path is the X.509 certification path that establishes the binding between the calling KMAE's distinguished name and the calling KMAE's key management public cryptographic material.

```
CallingCertPath ::= CertificationPath
```

3.4.2.3.3 Called certificate path

The Called Certificate Path is the X.509 certification path that establishes the binding between the called KMAE's distinguished name and the called KMAE's key management public cryptographic material.

```
CalledCertPath ::= CertificationPath
```

3.4.2.3.4 Calling key generation algorithm parameters

The Calling Key Generation Algorithm Parameters are the calling KMAE's parameters outside the certificate that may be present depending on the key generation algorithm used to create the key between the calling and called KMAE.

```
KEY-GEN-ALG.&Rq-Parms(@keyGenAlgorithmID) OPTIONAL
```

3.4.2.3.5 Called key generation algorithm parameters

The Called Key Generation Algorithm Parameters are the called KMAE's parameters outside the certificate that may be present depending on the key generation algorithm used to create the key between the calling and called KMAE.

```
KEY-GEN-ALG.&Rp-Parms(@keyGenAlgorithmID) OPTIONAL
```

3.4.2.3.6 Calling attribute certification path

The attribute certification path establishes a binding between the calling KMAE's distinguished name and a set of domain specific attributes. One example of attribute certification path can be found in ANSI X9.30. Use of attribute certification paths is optional.

```
AttributeCertificationPath ::= SEQUENCE {
      attrCert    AttributeCertificate,
      acPath      SEQUENCE OF ACPathData OPTIONAL }

ACPathData ::= SEQUENCE {
      cert        Certificate OPTIONAL,
      attrCert    AttributeCertificate OPTIONAL }
```

3.4.2.3.7 Called attribute certification path

The attribute certification path establishes a binding between the called KMAE's distinguished name and a set of domain specific attributes. Use of attribute certification paths is optional. The Called Attribute Certification Path and Calling Attribute Certification Path have the same syntax.

3.4.2.4 Send key (Send-Key)

The Send-Key service transfers symmetric cryptographic keying material from the local key management end system to the remote key management end system. The cryptographic keying material is enciphered in a KEK held by the remote KMAE. The cryptographic keying material may have been generated locally or obtained from a KDC or KTC. 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 Table 3-10.

Table 3-10—Send-Key parameters

Send-Key parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M	M (=)	3.4.1.1.2
KEK Identifier	M	M (=)	—	—	3.4.2.4.1
Request Parameters	M	M (=)	—	—	3.4.2.4.2
Response Parameters	—	—	M	M (=)	3.4.2.4.3
Result	—	—	M	M (=)	3.4.1.1.9

The remote KMAE must have previously obtained the KEK that it will use to decrypt the enciphered cryptographic keying material. If this material came from a KDC or KTC, the KEK may also come from the same center, through a manual distribution.

3.4.2.4.1 KEK identifier

The KEK Identifier names the previously distributed cryptographic keying material that was used to encrypt the enciphered cryptographic keying material within the Request Parameters. The KEK Identifier indirectly specifies the encryption algorithm that was used to protect the enciphered cryptographic keying material. The relationship between the KEK Identifier and the encryption algorithm is maintained in the SMIB.

```
KEKIdentifier ::= CHOICE {
      said        [0] CalledSAID,
      kdc         [1] KDCAETitle,
      ktc         [2] KTCAETitle }
```

3.4.2.4.2 Request parameters

The parameters sent to the remote key management end system vary depending upon the center protocol that is used. The center protocol is identified by an OBJECT IDENTIFIER. The associated parameters are registered with the center protocol OBJECT IDENTIFIER. A sample registration is presented in 3D.2 of Annex 3D.

The registered Request Parameters must include the Calling SAID assigned by the local key management end system and the package containing the symmetric cryptographic keying material enciphered in a KEK that was previously distributed to the remote key management end system.

```
centerProtocol CENTER-PROTOCOL.&centerProtocolId
sRqParms CENTER-PROTOCOL.&SendRqParms {@centerProtocol}
```

3.4.2.4.3 Response Parameters

The parameters returned by the remote key management end system vary depending upon the center protocol that is used. The center protocol is identified by an OBJECT IDENTIFIER in the Request Parameters. The associated parameters are registered with the center protocol OBJECT IDENTIFIER.

The registered Response Parameters must include the Calling SAID assigned by the local key management end system and the Called SAID assigned by the remote key management end system.

```
sRsParms CENTER-PROTOCOL.&SendRsParms {@centerProtocol}
```

3.4.2.5 Negotiate Security Association Attributes (Pick-SA-Attrs)

The Pick-SA-Attrs forms a security association by assigning attributes to cryptographic keying material that was established using the Select-Key, Make-Key, or Send-Key service. The same application-association used for Select-Key, Make-Key, Protected-Make-Key, Please-Send-Key, or Send-Key will be used to provide this service. This service synchronizes the enabling of cryptographic keying material and associated attributes within the SMIB for use by the Security Protocol through the second and third parts of a three-part exchange. This three-part security exchange is defined in 3.5.2.5. When the responding KMAE sends the second security exchange, the security protocol in that end system is prepared to receive protected traffic. When the initiating KMAE sends the third security exchange, the security protocol in that end system is prepared to send and receive protected traffic. When the responding KMAE receives the third security exchange, the security protocol in that end system can now send protected traffic.

Since the KMTI negotiation performed by Pick-KM-Alg is not integrity protected, the KMTI is repeated in Pick-SA-Attrs to detect malicious modification during Pick-KM-Alg.

The Pick-SA-Attrs exchange is protected by the cryptographic keying material that was established using the Select-Key, Make-Key, Send-Key, Please-Send-Key, or Protected-Make-Key service. This protection provides peer entity authentication, confidentiality, and integrity.

The parameters of Pick-SA-Attrs are defined in Table 3-11.

Table 3-11—Pick-SA-Attrs parameters

Pick-SA-Attrs parameter name	.req	.ind	.rsp	.conf	Reference
Security Association Attributes List	M	M (=)	—	—	3.4.1.1.5
Security AssociationAttributes	—	—	M	M (=)	3.4.1.1.6
Calling SAID	—	—	M	M (=)	3.4.1.1.7
Called SAID	M	M (=)	—	—	3.4.1.1.8
Key Management Technique Identifier	—	—	U	C (=)	3.4.1.1.3
Security Policy Identifier	—	—	U	C (=)	3.4.1.1.4
Key Recovery Info	U	C (=)	U	C (=)	3.4.2.5.1
Result	—	—	M	M (=)	3.4.1.1.9

3.4.2.5.1 Key recovery info

This optional parameter supports key recovery. The key recovery technique may be specified.

```
KeyRecoveryInfo ::= KeyRecoveryData OPTIONAL

KeyRecoveryData ::= SEQUENCE {
      recoveryTechnique OBJECT IDENTIFIER OPTIONAL,
      recoveryData      OCTET STRING }
```

3.4.2.6 Spawn key (Spawn-Key)

The Spawn-Key service forms a security association by transforming a previously established symmetric cryptographic key to form a new symmetric cryptographic key. Spawn-Key optionally assigns the attributes that were associated with the previously established key to the new 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.conf 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-Key exchange is protected by the previously established security association. This protection provides peer entity authentication; the protection also provides confidentiality, integrity, or both.

The parameters of Spawn-Key are defined in Table 3-12.

Table 3-12—Spawn-Key parameters

Spawn-Key parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M	M (=)	3.4.1.1.2
Key Transformation Algorithm Identifier	M	M (=)	—	—	3.4.1.2.2
Previously Established Calling SAID	M	M (=)	M	M (=)	3.4.1.2.3
Previously Established Called SAID	M	M (=)	—	—	3.4.1.2.4
Calling SAID	M	M (=)	M (=)	M (=)	3.4.1.1.7
Called SAID	—	—	M	M (=)	3.4.1.1.8
Result	—	—	M	M (=)	3.4.1.1.9

3.4.2.7 Get multicast key (Get-MKey)

The Get-MKey service 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 obtains the broadcast or multicast key as its needs dictate. The key center does not push the broadcast or multicast key out to every member of the group.

The Get-MKey service may only be used after a Select-Key, Make-Key, Send-Key, Please-Send-Key, or Protected-Make-Key service. The same application-association used for the Select-Key, Make-Key, Send-Key, Please-Send-Key, or Protected-Make-Key service will be used to provide this confirmed service. The security association identifier and all other security association attributes are assigned by the MKC 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.conf is provided, the cryptographic keying material and associated attributes have 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 authentication, integrity, and confidentiality; a security association that provides these security services must be used.

The parameters of Get-MKey are defined in Table 3-13.

Table 3-13—Get-MKey parameters

Get-MKey parameter name	.req	.ind	.rsp	.conf	Reference
Calling SAID	—	—	M	M (=)	3.4.1.1.7
Called SAID	M	M (=)	—	—	3.4.1.1.8
McastAddressList	M	M (=)	—	—	3.4.1.4.2
McastTokenList	—	—	M	M (=)	3.4.1.4.3
Result	—	—	M	M (=)	3.4.1.1.9

3.4.2.8 Delete key (Delete-Key)

The Delete-Key service notifies the remote key management end system that the cryptographic keying material and associated attributes, which define a particular security association, are no longer available. Deleting a security association may remove the keying material and associated attributes from the SMIB, if local policy permits. Delete-Key provides an unconfirmed service. The remote key management end system also makes unavailable the keying material and associated attributes that define the security association. Again, if local policy permits, deleting the security association may remove the keying material and associated attributes from the remote SMIB.

Portions of the Delete-Key exchange are protected by the security association that is to be deleted, providing data origin authentication. The protection may also include confidentiality, integrity, or both.

The parameters of Delete-Key are defined in Table 3-14.

Table 3-14—Delete-Key parameters

Delete-Key parameter name	.req	.ind	Reference
Calling AE-Title	M	M (=)	3.4.1.1.1
Called AE-Title	M	M (=)	3.4.1.1.2
Calling SAID	M	M (=)	3.4.1.1.7
Called SAID	M	M (=)	3.4.1.1.8

3.4.2.9 Release peer association (Release-P)

The Release-P service releases the application-association with the remote key management end system.

The parameters of Release-P are defined in Table 3-15.

Table 3-15—Release-P parameters

Release-P parameter name	.req	.ind	.rsp	.conf	Reference
Release-request-reason	U	C (=)	—	—	3.4.2.9.1
Release-response-reason	—	—	U	C (=)	3.4.2.9.2
User Information	U	C (=)	U	C (=)	3.4.2.9.3
Result	—	—	M	M (=)	3.4.1.1.9

3.4.2.9.1 Release-request-reason

The Release-request-reason parameter is used to indicate the reason for the release. Release request reason values are normal, urgent, or user-defined.

```
release-request-reason ::= INTEGER {
      normal(0),
      urgent(1),
      userdefined(30) }
```

3.4.2.9.2 Release-response-reason

The Release-response-reason parameter is used to indicate the reason for the release. Release response reason values are normal, not finished, or user-defined.

```
release-response-reason ::= INTEGER {
      normal(0),
      notfinished(1),
      userdefined(30) }
```

3.4.2.9.3 User information

The User Information provides additional information in regard to the release or abort. This information is not protected by any security service.

```
UserInformation ::= OCTET STRING
```

3.4.2.10 Abort peer association (Abort-P)

The Abort-P service aborts the application-association with the remote key management end system.

The parameters of Abort-P are defined in Table 3-16.

Table 3-16—Abort-P parameters

Abort-P parameter name	.req	.ind	Reference
Abort Source	—	M	3.4.2.10.1
User Information	U	C (=)	3.4.2.9.3

3.4.2.10.1 Abort source

The Abort Source value is used to indicate whether the abort was a service user abort (value equals zero) or a service provider abort (value equals one).

```
AbortSource ::= INTEGER {
      user(0),
      provider(1) }
```

3.4.2.11 Protected make key (Protected-Make-Key)

The Protected-Make-Key obtains the data needed to establish a symmetric cryptographic key using a public key cryptographic technique, but it does so under the protection of an existing security association. This service operates in the same manner as the Make-Key.

The parameters of Protected-Make-Key are defined in Table 3-17.

3.4.2.12 Get next multicast key (Get-Next-MKey)

The Get-Next-MKey service obtains the subsequent keying material for a new multicast security association. The service uses an existing multicast security association that must provide confidentiality.

The parameters of Get-Next-MKey are defined in Table 3-18.

Table 3-17—Protected-Make-Key parameters

Protected-Make-Key parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M	M (=)	3.4.1.1.2
Key Generation Algorithm Identifier	M	M (=)	—	—	3.4.2.3.1
Calling Certificate Path	M	M (=)	—	—	3.4.2.3.2
Called Certificate Path	—	—	M	M (=)	3.4.2.3.3
Calling Key Generation Algorithm Parameters	U	C (=)	—	—	3.4.2.3.4
Called Key Generation Algorithm Parameters	—	—	U	C (=)	3.4.2.3.5
Previously Established Calling SAID	M	M (=)	—	—	3.4.1.2.3
Previously Established Called SAID	M	M (=)	—	—	3.4.1.2.4
Calling SAID	M	M (=)	M (=)	M (=)	3.4.1.1.7
Called SAID	—	—	M	M (=)	3.4.1.1.8
Result	—	—	M	M (=)	3.4.1.1.9

Table 3-18—Get-Next-MKey parameters

Get-Next-MKey parameter name	.req	.ind	.rsp	.conf	Reference
McastSAID	M	M (=)	M (=)	M (=)	3.4.1.5.1
McastAddress	M	M (=)	M (=)	M (=)	3.4.1.4.2
McastToken	—	—	M	M (=)	3.4.1.4.3

3.4.2.13 Please send key (Please-Send-Key)

The Please-Send-Key primitive requests the remote key management end system to obtain a symmetric cryptographic key and send it to the local key management end system. The cryptographic keying material is enciphered in a KEK held by the local KMAE; the cryptographic keying material may have been obtained from the KDC or the KTC. If there is not an application-association with the remote key management end system, then one is created.

Please-Send-Key provides an unconfirmed service, and the remote key management end system is expected to initiate the Send-Key confirmed service upon receipt of the Please-Send-Key indication.

The parameters of Please-Send-Key are defined in Table 3-19.

Table 3-19—Please-Send-Key parameters

Please-Send-Key parameter name	.req	.ind	Reference
Calling AE-Title	M	M (=)	3.4.1.1.1
Called AE-Title	M	M (=)	3.4.1.1.2
KEK Identifier	M	M (=)	3.4.2.4.1

3.4.3 Key center application service object (KCASO) services

The KCASO provides four services: Request-Key, Translate-Key, Release-C, and Abort-C.

3.4.3.1 Request key (Request-Key)

The Request-Key service obtains symmetric cryptographic keying material from a KDC. Two copies of the symmetric cryptographic keying material are obtained; one is enciphered in a KEK that was previously distributed to the local key management end system, and the other is enciphered in a KEK that was previously distributed to the remote key management end system. Request-Key provides a confirmed service. If there is not an application-association with the KDC, then one is created. Request-Key does not require an application-association with the remote key management end system.

The parameters of Request-Key are defined in Table 3-20.

Table 3-20—Request-Key parameters

Request-Key parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
KDC AE-Title	M	M (=)	M	M (=)	3.4.3.1.1
Called AE-Title	M	M (=)	M (=)	M (=)	3.4.1.1.2
KDC Request Parameters	M	M (=)	—	—	3.4.3.1.2
KDC Response Parameters	—	—	M	M (=)	3.4.3.1.3
Result	—	—	M	M (=)	3.4.1.1.9

3.4.3.1.1 KDC AE-Title

The KDC AE-Title comprises the Application Title and Application Entity Qualifier of the called KMAE, both of which are necessary during the application-association establishment and passed from the calling key management application to indicate the intended KDC. The form of the AE-Title could either be the distinguished name of the application or an object identifier for the application.

```
KDCAETitle ::= AETitle
```

3.4.3.1.2 KDC request parameters

The parameters sent to the KDC vary depending upon the center protocol that is used. The center protocol is identified by an OBJECT IDENTIFIER. The associated parameters are registered with the center protocol OBJECT IDENTIFIER. A sample registration is presented in 3D.2 of Annex 3D.

```
centerProtocol CENTER-PROTOCOL.&centerProtocolId
rRqParms CENTER-PROTOCOL.&RequestRqParms
                  {@centerProtocol}
```

3.4.3.1.3 KDC response parameters

The parameters returned by the KDC vary depending upon the center protocol that is used. The center protocol is identified by an OBJECT IDENTIFIER in the Request Parameters. The associated parameters are registered with the center protocol OBJECT IDENTIFIER.

The registered Response Parameters must accommodate two possible cases. When the KDC is unable to fulfill the request, the response contains either a referral to another KDC to handle the request, or an error. When the KDC is able to fulfill the request, the response contains two packages. One package contains the symmetric cryptographic keying material enciphered in a KEK that was previously distributed to the local key management end system, and the other package contains the symmetric cryptographic keying material enciphered in a KEK that was previously distributed to the remote key management end system. The second package will be sent to the remote key management end system as part of the Send-Key service response.

```
rRsParms CENTER-PROTOCOL.&RequestRsParms
                  {@centerProtocol}
```

3.4.3.2 Translate key (Translate-Key)

The Translate-Key service sends symmetric cryptographic keying material that was enciphered using a KEK that is shared between the key local management end system and the KTC, and obtains the same symmetric cryptographic keying material encrypted in a KEK that is shared between the remote key management end system and the KTC. Translate-Key provides a confirmed service. If there is not an application-association with the KTC, then one is created. Translate-Key does not require an application-association with the remote key management end system.

The parameters of Translate-Key are defined in Table 3-21.

Table 3-21—Translate-Key parameters

Translate-Key parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
KTC AE-Title	M	M (=)	M	M (=)	3.4.3.2.1
KTC Request Parameters	M	M (=)	—	—	3.4.3.2.2
KTC Response Parameters	—	—	M	M (=)	3.4.3.2.3
Result	—	—	M	M (=)	3.4.1.1.9

3.4.3.2.1 KTC AE-Title

The KTC AE-Title comprises the Application Title and Application Entity Qualifier of the called KMAE, both of which are necessary during the application-association establishment and passed from the calling key management application to indicate the intended KTC. The form of the AE-Title could either be the distinguished name of the application or an OBJECT IDENTIFIER.

```
KTCAETitle ::= AETitle
```

3.4.3.2.2 KTC request parameters

The parameters sent to the KTC vary depending upon the center protocol that is used. The center protocol is identified by an OBJECT IDENTIFIER. The associated parameters are registered with the center protocol OBJECT IDENTIFIER. A sample registration is presented in 3D.2 of Annex 3D.

```
centerProtocol CENTER-PROTOCOL.&centerProtocolId
tRqParms CENTER-PROTOCOL.&TranslateRqParms
                  {@centerProtocol}
```

3.4.3.2.3 KTC response parameters

The parameters returned by the KTC vary depending upon the center protocol that is used. The center protocol is identified by an OBJECT IDENTIFIER in the Request Parameters. The associated parameters are registered with the center protocol OBJECT IDENTIFIER.

The registered Response Parameters must accommodate two possible cases. When the KTC is unable to fulfill the request, the response contains either a referral to another KTC to handle the request, or an error. When the KTC is able to fulfill the request, the response contains one package that holds the symmetric cryptographic keying material enciphered in a KEK that was previously distributed to the remote key management end system. The package will be sent to the remote key management end system as part of the Send-Key service response.

```
tRsParms CENTER-PROTOCOL.&TranslateRsParms
                 {@centerProtocol}
```

3.4.3.3 Release center association (Release-C)

The Release service releases the application-association with the KDC or the KTC.

The parameters of Release-C are defined in Table 3-22.

Table 3-22—Release-C parameters

Release-C parameter name	.req	.ind	.rsp	.conf	Reference
Release-request-reason	U	C (=)	—	—	3.4.2.9.1
Release-response-reason	—	—	U	C (=)	3.4.2.9.2
User Information	U	C (=)	U	C (=)	3.4.2.9.3
Result	—	—	M	M (=)	3.4.1.1.9

3.4.3.4 Abort center association (Abort-C)

The Abort service aborts the application-association with the KDC or the KTC.

The parameters of Abort-C are defined in Table 3-23.

Table 3-23—Abort-C parameters

Abort-C parameter name	.req	.ind	Reference
Abort Source	M	—	3.4.2.10.1
User Information	U	C (=)	3.4.2.9.3

3.5 Security exchanges

This subclause defines the security exchanges that implement the KMP.

```
IMPORTS
    PROTECTED, PROTECTED-Q, SECURITY-EXCHANGE,
    SECURITY-TRANSFORMATION, PROTECTION-MAPPING,
    enciphered, signed, optionally-signed
       FROM Notation
         {joint-iso-ccitt genericULS(20) modules(1) notation(1)}
    CertificationPath, AlgorithmIdentifier
       FROM AuthenticationFramework
         {joint-iso-ccitt ds(5) module(1) authenticationFramework(7) 2}
```

3.5.1 Key management application entity (KMAE) security exchanges

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

3.5.2 Key peer application service object (KPASO) security exchanges

3.5.2.1 Negotiate key management algorithm (Pick-KM-Alg) security exchange

```
pick-km-alg SECURITY-EXCHANGE ::=
{
      SE-ITEMS    {pick-km-alg-1, pick-km-alg-2}
      IDENTIFIER  {se-id-pick-km-alg (1) }
}
pick-km-alg-1 SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   PickKMAlgRq
      ERRORS      {alg-negotiation-failed}
      ITEM-ID     1
}
pick-km-alg-2 SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   PickKMAlgRs
      ITEM-ID     2
}
alg-negotiation-failed SE-ERROR  ::=
{
      PARAMETER   Result
      ERROR-CODE  1
}
PickKMAlgRq::= SEQUENCE  {
                  KMTechniqueIdList OPTIONAL,
                  SecurityPolicy OPTIONAL  }

PickKMAlgRs::= SEQUENCE  {
                  KMTechniqueId OPTIONAL,
                  SecurityPolicy OPTIONAL  }

KMTechniqueIdList::= SEQUENCE OF KMTechniqueId
```

```
KMTechniqueId::= SEQUENCE  {
      technique          Technique,
      keyCreationAlg     KeyCreationAlg,
      attrConfidAlg      AttrConfidAlg,
      attrIntegrityAlg   AttrIntegrityAlg,
      transformationAlg TransformationAlg OPTIONAL  }

Technique::= INTEGER  {
                  manual(0),
                  requestorCallsCenter(1),
                  responderCallsCenter(2),
                  certificate(3)  }

KeyCreationAlg    ::= AlgorithmIdentifier
AttrConfidAlg     ::= AlgorithmIdentifier
AttrIntegrityAlg  ::= AlgorithmIdentifier
TransformationAlg ::= AlgorithmIdentifier

SecurityPolicy    ::= SEQUENCE {
      policyId            OBJECT IDENTIFIER,
      policyVersion       INTEGER OPTIONAL,
      authenticationInfo  OCTET STRING OPTIONAL  }

Result ::= INTEGER {
success(0),
-- The service has completed successfully.
-- General Error Codes
failed_no_reason(1),
-- The service did not complete successfully with no reason given.
service_not_available(2),
-- The underlying service provider is not available.
service_not_available_at_this_time(3),
-- The underlying service is not available at the particular moment and
-- the caller is encouraged to try again at a later time.
illegal_ASN1_value_bad_value(4),
-- The ASN values passed from the calling party contain a bad value and
-- the call is rejected.
illegal_ASN1_value_missing_field(5),
-- The ASN values passed from the calling party do not contain fields
-- that are mandatory and the call is rejected.
illegal_ASN1_value_unknown_field(6),
-- The valid ASN values passed from the calling party contain an unknown
-- field and the call is rejected.
provider_abort_received(7),
-- The underlying service provider aborted causing the service to fail.
user_abort_received(8),
-- The user service on the local side initiated an abort causing the
-- service to fail.
-- KMP Specific Error Codes
negotiation_failure(64),
-- The optional negotiation phase (Pick-KM-Algs) was not successful.
error_generating_key_material(65),
-- The key generation process (Select-Key, Send-Key, Make-Key, or
-- Protected-Make-Key) could not successfully generate the new keying
-- material.
error_retrieving_key_material(67),
-- The key retrieval process (Request-Key or Translate-Key)
-- could not successfully retrieve the new keying material from the key
-- center.
not_ready_for_protected_service(68),
-- Even though the key generation process completed successfully, the
-- protective security mechanism was not ready.
error_retrieving_multicast_key(69),
-- The key retrieval process (Get-MKey or Get-Next-Mkey) could not
-- successfully retrieve the new multicast key from the MKC.
unknown_security_transformation(70),
-- The security transformation was unknown during the Spawn-Key service.
error_processing_SA_attributes(71),
-- The attribute negotiation process (Pick-SA-Attrs) was not successful.
invalid_called_SAID(72),
-- The called SAID was invalid.
invalid_calling_SAID(73),
-- The calling SAID was invalid.
release_peer_urgent(74),
-- The KPASO generated an urgent release request reason.
release_peer_user_defined_rq(75),
-- The KPASO generated an user defined release request reason.
release_peer_not_finished(76),
-- The KPASO generated a not finished release response reason.
release_peer_user_defined_rp(77),
-- The KPASO generated an user defined release response.
release_center_urgent(78),
-- The KCASO generated an urgent release request reason.
release_center_user_defined_rq(79),
-- The KCASO generated a user defined release request reason.
release_center_not_finished(80),
-- The KCASO generated a not finished release response reason.
release_center_user_defined_rp(81),
-- The KCASO generated a user defined release response.
abort_peer_user(82),
-- The KPASO generated an user abort.
abort_peer_provider(83),
-- The KPASO generated a provider abort.
abort_center_user(84),
-- The KCASO generated an user abort.
abort_center_provider(85),
-- The KCASO generated a provider abort.
expired_key(86),
-- The Select-Key request referenced keying material that has expired.
unavailable_key(87),
-- The referenced keying material is not available at this
time.transform_count(256-511)
-- Contains the number of transformations to the base keying material
-- needed derive shared keying material. To determine this value,
-- subtract 256 from the error code received.
      }
```

3.5.2.2 Select key (Select-Key) security exchange

```
select-key  SECURITY-EXCHANGE ::=
{
      SE-ITEMS    {select-key-1, select-key-2}
      IDENTIFIER  {se-id-select-key (2)}
}
select-key-1  SEC-EXCHG-ITEM ::=
{
      ITEM-TYPE   SelectKeyRq
      ERRORS      {select-key-failed}
      ITEM-ID     1
}
select-key-2  SEC-EXCHG-ITEM  :=
{
      ITEM-TYPE   SelectKeyRs
      ITEM-ID     2
}
select-key-failed SE-ERROR  ::=
{
      PARAMETER   Result
      ERROR-CODE  2
}

SelectKeyRq::=  SEQUENCE  {
            KeyingMaterialIdentifier,
            KeyTransformationAlgorithmIdentifier OPTIONAL,
            AlgorithmIdentifier, -- used to encrypt following sequence
            PROTECTED  { SEQUENCE  {
                  CallingSAID,
                  Nonce
            }, enciphered  }  }

SelectKeyRs  ::= PROTECTED  {  SEQUENCE  {
                  CallingSAID,
                  CalledSAID,
                  NoncePlus1
            }, enciphered  }

CallingSAID ::= OCTET STRING
CalledSAID  ::= OCTET STRING
KeyingMaterialIdentifier  ::= OCTET STRING
KeyTransformationAlgorithmIdentifier ::= AlgorithmIdentifier
Nonce  ::= INTEGER
NoncePlus1  ::= Nonce
```

3.5.2.3 Make key (Make-Key) security exchange

```
make-key    SECURITY-EXCHANGE  ::=
{
      SE-ITEMS    {make-key-1, make-key-2}
      IDENTIFIER  {se-id-make-key (3)}
}
make-key-1  SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   MakeKeyRq
      ERRORS      {make-key-failed}
      ITEM-ID     1
}
make-key-2  SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   MakeKeyRs
      ITEM-ID     2
}
make-key-failed SE-ERROR  ::=
{
      PARAMETER   Result
      ERROR-CODE  3
}
MakeKeyRq::=  SEQUENCE  {
      keyGenAlgorithmID KEY-GEN-ALG.&key-Gen-Alg-Id,
                        CallingCertPath,
                        AttributeCertificationPath OPTIONAL,
                        KEY-GEN-ALG.&Rq-Parms { @keyGenAlgorithmID }
                                          OPTIONAL,
                        CallingSAID  }

MakeKeyRs::= SEQUENCE  {
                        CalledCertPath,
                        AttributeCertificationPath OPTIONAL,
                        KEY-GEN-ALG.&Rp-Parms { @keyGenAlgorithmID }
                                          OPTIONAL,
                        CallingSAID,
                        CalledSAID  }

CallingCertPath   ::= CertificationPath
CalledCertPath    ::= CertificationPath
AttributeCertificationPath  ::= SEQUENCE {
    attrCert      AttributeCertificate,
    acPath        SEQUENCE OF ACPathData OPTIONAL }

ACPathData  ::= SEQUENCE {
    cert          Certificate OPTIONAL,
    attrCert      AttributeCertificate OPTIONAL }
```

3.5.2.4 Send key (Send-Key) security exchange

```
send-key    SECURITY-EXCHANGE  ::=
{
      SE-ITEMS    {send-key-1, send-key-2}
      IDENTIFIER  {se-id-send-key (4)}
}
send-key-1  SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   SendKeyRq
      ERRORS      {send-key-failed}
      ITEM-ID     1
}
```

```
send-key-2  SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   SendKeyRs
      ITEM-ID     2
}
send-key-failed   SE-ERROR  ::=
{
      PARAMETER   Result
      ERROR-CODE  4
}
SendKeyRq::=  SEQUENCE  {
      kekIdentifier     KEKIdentifier,
      centerProtocol    CENTER-PROTOCOL.&centerProtocolId,
      sRqParms          CENTER-PROTOCOL.&SendRqParms {@centerProtocol} }

SendKeyRs::=  SEQUENCE  {
      sRsParms          CENTER-PROTOCOL.&SendRsParms {@centerProtocol} }

KEKIdentifier     ::= CHOICE  {
      said        [0]  CalledSAID,
      kdc         [1]  KDCAETitle,
      ktc         [2]  KTCAETitle  }

KDCAETitle        ::=  AETitle
KTCAETitle        ::=  AETitle

AETitle::=  CHOICE  {
                  AETitle-form1,
                  AETitle-form2  }

AETitle-form1     ::=  [0]  SEQUENCE  {  {
                  COMPONENTS OF
                  APTitle-form1,
                  AEQualifier-form1  }  }

APTitle-form1     ::=  IMPLICIT SEQUENCE OF RelativeDistinguishedName

APQualifier-form1 ::=  [1]  IMPLICIT RelativeDistinguishedName

AETitle-form2     ::=  [1]  SEQUENCE  {
                  APTitle-form2,
                  AEQualifier-form2 }

APTitle-form2     ::= [2]  IMPLICIT OBJECT IDENTIFIER

AEQualifier-form2 ::= [3]  IMPLICIT INTEGER
```

3.5.2.5 Negotiate security association attributes (Pick-SA-Attrs) security exchange

```
pick-sa-attrs     SECURITY-EXCHANGE  ::=
{
      SE-ITEMS    {pick-sa-attrs-1, pick-sa-attrs-2, pick-sa-attrs-3}
      IDENTIFIER  {se-id-pick-sa-attrs (5)}
}
```

```
pick-sa-attrs-1  SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   PickSAAttrsRq
      ERRORS      {attrs-negotiation-failed}
      ITEM-ID     1
}
pick-sa-attrs-2  SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   PickSAAttrsRs
      ERRORS      {attrs-negotiation-failed}
      ITEM-ID     2
}
pick-sa-attrs-3  SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   PickSAAttrsCommit
      ITEM-ID     3
}
attrs-negotiation-failed SE-ERROR  ::=
{
      PARAMETER   Result
      ERROR-CODE  5
}

PickSAAttrsRq::= SEQUENCE  {
    CalledSAID,
    KeyRecoveryData OPTIONAL,
    PROTECTED-Q      { SAAttrsList,
      sealedEncrypt, { attrConfidAlg, attrIntegrityAlg, calledSAID } } }

PickSAAttrsRs::=  SEQUENCE  {
    CallingSAID,
    KeyRecoveryData OPTIONAL,
    PROTECTED-Q      { NegotiatedAttrs,
      sealedEncrypt, { attrConfidAlg, attrIntegrityAlg, callingSAID } } }

PickSAAttrsCommit  ::= SEQUENCE  {
    CalledSAID,
    PROTECTED-Q { Commit,
      sealedEncrypt, { attrConfidAlg, attrIntegrityAlg, calledSAID } } }

NegotiatedAttrs  ::=  SEQUENCE  {
      SAAttrs,
      [0]  KMTechniqueId OPTIONAL,
      [1]  SecurityPolicy OPTIONAL  }

SAAttrsList::=  SEQUENCE OF SAAttrs

SAAttrs::=  CHOICE  {
      sameAs      [0]   ReferenceSA,
                  [1]   SPAttrs  }

ReferenceSA::=  SEQUENCE  {
      callingSAID CallingSAID,
      calledSAID  CalledSAID,
      padding     OCTET STRING OPTIONAL  }

SPAttrs::=  SEQUENCE  {
      spAttrs     SP-ATTRS.&attrs-Id,
      attrs       SP-ATTRS.&Sp-Attrs {@spAttrs} OPTIONAL  }

Commit::=  SEQUENCE  {
            Nonce,
            Result  }

KeyRecoveryData  ::= SEQUENCE  {
      recoveryTechnique OBJECT IDENTIFIER OPTIONAL,
      recoveryData      OCTET STRING  }
```

3.5.2.6 Spawn key (Spawn-Key) security exchange

```
spawn-key   SECURITY-EXCHANGE  ::=
{
      SE-ITEMS    { spawn-key-1, spawn-key-2}
      IDENTIFIER  {se-id-spawn-key (6)}
}
spawn-key-1 SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   SpawnKeyRq
      ERRORS      {spawn-key-failed}
      ITEM-ID     1
}
spawn-key-2 SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   SpawnKeyRs
      ITEM-ID     2
}
spawn-key-failed  SE-ERROR  ::=
{
      PARAMETER   Result
      ERROR-CODE  6
}
SpawnKeyRq  ::=  SEQUENCE  {
            PreviouslyEstablishedCalledSAID,
            PROTECTED  {  SEQUENCE  {
                  KeyTransformationAlgorithmIdentifier,
                  PreviouslyEstablishedCallingSAID,
                  CallingSAID
            },  enciphered  }  }

SpawnKeyRs  ::=  SEQUENCE  {
            PreviouslyEstablishedCallingSAID,
            PROTECTED  {  SEQUENCE  {
                  CallingSAID,
                  CalledSAID
            }, enciphered  }  }

PreviouslyEstablishedCallingSAID::=  OCTET STRING
PreviouslyEstablishedCalledSAID::=  OCTET STRING
```

3.5.2.7 Get multicast key (Get-MKey) security exchange

```
get-mkey    SECURITY-EXCHANGE  ::=
{
      SE-ITEMS    {get-mkey-1, get-mkey-2}
      IDENTIFIER  {se-id-get-mkey (7)}
}
get-mkey-1  SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   GetMkeyRq
      ERRORS      {get-mkey-failed}
      ITEM-ID     1
}
get-mkey-2  SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   GetMkeyRs
      ITEM-ID     2
}
get-mkey-failed  SE-ERROR  ::=
{
      PARAMETER   Result
      ERROR-CODE  7
}
GetMkeyRq::=  SEQUENCE  {
      CalledSAID,
      PROTECTED  { McastAddressList,
                        sealedEncrypt  }   }

GetMkeyRs::=  SEQUENCE  {
      CallingSAID,
      PROTECTED   { McastTokenList,
                        sealedEncrypt  }  }

McastAddressList  ::=  SEQUENCE OF McastAddress
McastAddress      ::=  OCTET STRING
McastTokenList    ::=  SEQUENCE OF McastToken
McastToken        ::=  SEQUENCE  {
      mKCId             MKCTitle,
                        PROTECTED  {  SEQUENCE  {
      mcastKey                OCTET STRING,
      mcastAddress            OCTET STRING,
      mcastAttributes         SPAttrs,
      mcastSAID               McastSAID,
      validityPeriod          ValidityPeriod  }, optionally-signed  }  }

ValidityPeriod    ::=  SEQUENCE  {
      notBefore   GeneralizedTime,
      notAfter    GeneralizedTime  }

MKCTitle          ::=  AETitle
McastSAID         ::=  OCTET STRING
```

3.5.2.8 Delete key (Delete-Key) security exchange

```
delete-keySECURITY-EXCHANGE  ::=
{
      SE-ITEMS    {delete-key-1}
      IDENTIFIER  {se-id-delete-key (8)}
}
delete-key-1      SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   DeleteKeyRq
      ITEM-ID     1
}
DeleteKeyRq  ::=  SEQUENCE  {
            CalledSAID,
            PROTECTED  {  SEQUENCE  {
                  Nonce,
                  CallingSAID
            },  enciphered  }  }
```

3.5.2.9 Protected make key (Protected-Make-Key)

```
protected-make-key  SECURITY-EXCHANGE  ::=
{
      SE-ITEMS    {protected-make-key-1, protected-make-key-2}
      IDENTIFIER  {se-id-protected-make-key (12)}
}
protected- make-key-1  SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   ProtectedMakeKeyRq
      ERRORS      {protected-make-key-failed}
      ITEM-ID     1
}
protected- make-key-2  SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   ProtectedMakeKeyRs
      ITEM-ID     2
}
protected-make-key-failed  SE-ERROR ::=
{
      PARAMETER   Result
      ERROR-CODE  12
}
ProtectedMakeKeyRq  ::= SEQUENCE  {
            CallingSAID,
            PROTECTED  {  MakeKeyRq,  sealedEncrypt  }  }

ProtectedMakeKeyRs  ::= SEQUENCE  {
            CalledSAID,
            PROTECTED  {  MakeKeyRs,  sealedEncrypt  }  }
```

3.5.2.10 Get next multicast key (Get-Next-MKey)

```
get-next-mkey      SECURITY-EXCHANGE  ::=
{
      SE-ITEMS    {get-next-mkey-1, get-next-mkey-2}
      IDENTIFIER  {se-id-get-next-mkey (13)}
}
get-next-mkey-1   SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   GetNextMkeyRq
      ERRORS      {get-next-mkey-failed}
      ITEM-ID     1
}
get-next-mkey-2   SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   GetNextMkeyRs
      ITEM-ID     2
get-next-mkey-failed  SE-ERROR  ::=
{
      PARAMETER   Result
      ERROR-CODE  13
}
GetNextMkeyRq::=  SEQUENCE  {
            McastSAID,
            PROTECTED  {  McastAddress, sealedEncrypt  }  }

GetNextMkeyRs::=  SEQUENCE  {
            McastSAID,
            PROTECTED  {  McastToken, sealedEncrypt  }  }
```

3.5.2.11 Please send key (Please-Send-Key) security exchange

```
please-send-key   SECURITY-EXCHANGE  ::=
{
      SE-ITEMS    {please-send-key-1, please-send-key-2,
                        please-send-key-3}
      IDENTIFIER  {se-id-please-send-key (11)}
}
please-send-key-1  SEC-EXCHG-ITEM  ::=
{
      ITEM TYPE   PleaseSendKeyRq
      ERRORS      {please-send-key-failed}
      ITEM ID     1
}
please-send-key-2  SEC-EXCHG-ITEM  ::=
{
      ITEM TYPE   SendKeyRq
      ERRORS      {send-key-failed}
      ITEM ID     2
}
please-send-key-3  SEC-EXCHG-ITEM  ::=
{
      ITEM TYPE   SendKeyRs
      ITEM ID     3
}
please-send-key-failed  SE-ERROR  ::=
{
      PARAMETER   Result
      ERROR CODE  11
}
PleaseSendKeyRq::=  SEQUENCE  {
            KEKIdentifier  }
```

3.5.3 Key center application service object (KCASO) security exchanges

3.5.3.1 Request key (Request-Key) security exchange

```
request-key SECURITY-EXCHANGE  ::=
{
      SE-ITEMS    {request-key-1, request-key-2}
      IDENTIFIER  {se-id-request-key (9)}
}
request-key-1     SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   RequestKeyRq
      ERRORS      {request-key-failed}
      ITEM-ID     1
}
request-key-2     SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   RequestKeyRs
      ITEM-ID     2
}
request-key-failed  SE-ERROR  ::=
{
      PARAMETER   Result
      ERROR-CODE  9
}
RequestKeyRq            ::=  SEQUENCE  {
      callingAETitle    CallingAETitle,
      centerProtocol    CENTER-PROTOCOL.&centerProtocolId,
      rRqParms    CENTER-PROTOCOL.&RequestRqParms {@centerProtocol} }

RequestKeyRs            ::=  SEQUENCE  {
      kdcAETitle        KDCAETitle,
      rRsParms    CENTER-PROTOCOL.&RequestRsParms {@centerProtocol} }

CallingAETitle          ::=  AETitle
CalledAETitle           ::=  AETitle
```

3.5.3.2 Translate key (Translate-Key) security exchange

```
translate-key     SECURITY-EXCHANGE  ::=
{
      SE-ITEMS    {translate-key-1, translate-key-2}
      IDENTIFIER  {se-id-translate-key (10)}
}
```

```
translate-key-1   SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   TranslateKeyRq
      ERRORS      {translate-key-failed}
      ITEM-ID     1
}
translate-key-2   SEC-EXCHG-ITEM  ::=
{
      ITEM-TYPE   TranslateKeyRs
      ITEM-ID     2
}
translate-key-failed  SE-ERROR  ::=
{
      PARAMETER   Result
      ERROR-CODE  10
}
TranslateKeyRq          ::=  SEQUENCE  {
      callingAETitle    CallingAETitle,
      centerProtocol    CENTER-PROTOCOL.&centerProtocolId,
      tRqParms    CENTER-PROTOCOL.&TranslateRqParms {@centerProtocol} }

TranslateKeyRs          ::=  SEQUENCE  {
      ktcAETitle        KTCAETitle,
      tRsParms    CENTER-PROTOCOL.&TranslateRsParms {@centerProtocol} }

KTCAETitle::=  AETitle
```

3.5.4 Object identifiers

```
sils  ::= { iso (1) member-body (2) us (840) ieee-802dot10 (10022) }
id-kmp-se               ::= { sils 11 }
id-kmp-center-exchange  ::= { sils 22 }

se-id-pick-km-alg             ::=  { id-kmp-se 1 }
se-id-select-key              ::=  { id-kmp-se 2 }
se-id-make-key                ::=  { id-kmp-se 3 }
se-id-send-key                ::=  { id-kmp-se 4 }
se-id-pick-sa-attrs           ::=  { id-kmp-se 5 }
se-id-spawn-key               ::=  { id-kmp-se 6 }
se-id-get-mkey                ::=  { id-kmp-se 7 }
se-id-delete-key              ::=  { id-kmp-se 8 }
se-id-request-key             ::=  { id-kmp-se 9 }
se-id-translate-key           ::=  { id-kmp-se 10 }
se-id-please-send-key         ::=  { id-kmp-se 11 }
se-id-protected-make-key      ::=  { id-kmp-se 12 }
se-id-get-next-mkey           ::=  { id-kmp-se 13 }
se-id-cert-replacement        ::=  { id-kmp-se 14 }
se-id-request-cml             ::=  { id-kmp-se 15 }
```

3.5.5 Object class definitions

3.5.5.1 Key generation algorithm object class

This subclause defines a class of information objects.

```
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
```

3.5.5.2 Security protocol attributes object class

```
SP-ATTRS    ::= CLASS
{
&attrs-Id         OBJECT IDENTIFIER,
                  &Sp-Attrs
}
WITH SYNTAX
{
      SP-ATTRS    &Sp-Attrs
      ATTRS-ID    &attrs-Id
}
```

3.5.5.3 Center protocol object class

```
CENTER-PROTOCOL::= CLASS
{
      &centerProtocolId       OBJECT IDENTIFIER
                              & SendRqParms
                              & SendRsParms
}
WITH SYNTAX
{
      SEND-RQ-PARMS           & SendRqParms
      SEND-RS-PARMS           & SendRsParms
      CENTER-PROTOCOL-ID      &centerProtocolId
}
```

3.5.6 Security transformations and protection mappings

This standard uses the security transformations and protection mappings defined in the GULS standard wherever possible. However, where these are not sufficient, new transformations and mappings are used. This subclause defines these additional security transformations and protection mappings.

3.5.6.1 Integrity and privacy security transformation

```
integrityAndPrivacyTransform  SECURITY-TRANSFORMATION ::=
{
      IDENTIFIER              {silsSecurityTransformation
                              sealed-enciphered(1)}
      INITIAL-ENCODING-RULES  {joint-iso-ccitt asn1(1) ber(1)}
      XFORMED-DATA-TYPE       PROTECTED {
                              PROTECTED-Q {
                                     ToBeProtected, sealed, intAlg},
                              enciphered}
      QUALIFIER-TYPE          Qtype
}
Qtype       ::=  SEQUENCE  {
intAlg            AlgorithmIdentifier,
confAlg           AlgorithmIdentifier,
sa-id             SAID  }
```

3.5.6.1.1 Other details

Other details of the Integrity and Privacy Security transform are as follows:

— *Encoding process and its local inputs*. The Base-type has an ICV calculated and appended and then the entire sequence is encrypted.

— *Decoding process and local inputs/outputs*. The transformed type is decrypted and the ICV is calculated and checked.

— *Parameters*. All parameters are supplied with the algorithm identifiers.

— *Transformation qualifiers*. The algorithm identifier and SAID identify the keying material.

— *Errors*. On decoding, the ICV calculation will fail.

— *Security services*. Integrity and confidentiality.

3.5.6.2 Integrity security transformation

```
integrityTransform            SECURITY-TRANSFORMATION ::=
{
      IDENTIFIER              {silsSecurityTransformation sealed (2)}
      INITIAL-ENCODING-RULES  {joint-iso-ccitt asn1(1) ber(1)}
      XFORMED-DATA-TYPESEQUENCE { ToBeProtected, Icv}
      QUALIFIER-TYPE          Itype
}
Itype ::=   AlgorithmIdentifier
Icv   ::=   OCTET STRING
```

3.5.6.3 Seal and encrypt protection mapping

```
sealedEncrypt     PROTECTION-MAPPING  ::=
{
      SECURITY-TRANSFORMATION
            {integrityAndPrivacyTransform}
}
```

3.5.6.4 Sealed protection mapping

```
sealed      PROTECTION-MAPPING  ::=
{
      SECURITY-TRANSFORMATION       {integrityTransform}
}
```

3.6 KMAE control function

This subclause describes the KMAE Control Function.

3.6.1 KMAE control function state tables

This subclause specifies the state tables for the KMAE CF. The state tables (Tables 3-24 through 3-34) do not include interactions with ACSE; rather, they assume that the CF establishes application-associations when needed and releases them when no longer needed.

Each state table has state names across the top as column headings, and event names down the first column as row headings. Within each state table, cells that are not blank contain information that the protocol machine uses to generate an action and then to transition to a new state. This information includes predicates, which the protocol machine tests, and if true performs the action indicated and the transition to the state indicated. If a predicate is not present, the protocol machine unconditionally performs the action and transitions to the new state. Within a cell, P indicates predicate, A indicates action, and S indicates resulting state.

Within each state table, every cell that is blank is invalid and causes an Abort.req service request, and results in an Idle state. Likewise, the receipt of an Abort.ind causes the protocol machine to transition to the Idle state.

Table 3-24—Create-SA REQUESTER

Event	State								
	Idle	**Wait-KM-Alg.rsp**	**Wait-Request.conf**	**Wait-Translate. conf**	**Wait-Send.conf**	**Wait-Make.conf**	**Wait-Select.conf**	**Wait-Attrs**	**Wait-Send.ind**
Create-SA.req	**P:**(KMTI List) > 1 or “must announce security policy” **A:** Pick-KM-Alg.req **S:** Wait-KM-Alg.rsp else **P:** technique = 0 **A:** Select-Key.req **S:** Wait-select.conf else **P:** technique = 3 **A:** Make-Key.req **S:** Wait-Make.conf else **P:** technique = 2 **A:** Please-Send-Key.req **S:** Wait-Send.ind else **P:** Local Key Gen allowed = Yes **A:** Translate-Key.req **S:** Wait-Translate.conf else **A:** Request-Key.req **S:** Wait-Request.conf								

Table 3-24—Create-SA REQUESTER (continued)

Event	**State**								
	Idle	**Wait-KM-Alg.rsp**	**Wait-Request.conf**	**Wait-Translate. conf**	**Wait-Send.conf**	**Wait-Make.conf**	**Wait-Select.conf**	**Wait-Attrs**	**Wait-Send.ind**
Pick-KM-Alg.conf		**P:** technique = 0 **A:** Select-Key.req **S:** Wait-Select.conf else **P:** technique = 3 **A:** Make-Key.req **S:** Wait-Make.conf else **P:** technique = 2 **A:** Please-Send-Key.req **S:** Wait-Send.ind else **P:** local key gen allowed = Yes **A:** Translate-Key.req **S:** Wait-Translate.conf else **A:** Request-Key.req **S:** Wait-Request.conf							
Request-Key.conf			**P:** Package returned? **A:** Send-Key.req **S:** Wait-Send.conf else **A:** Translate-Key.req **S:** Wait-Translate.conf						

Table 3-24—Create-SA REQUESTER (continued)

Event	**State**								
	Idle	**Wait-KM-Alg.rsp**	**Wait-Request.conf**	**Wait-Translate. conf**	**Wait-Send.conf**	**Wait-Make.conf**	**Wait-Select.conf**	**Wait-Attrs**	**Wait-Send.ind**
Translate-Key.conf				**P:** Package returned? **A:** Send-Key.req **S:** Wait-Send.conf else **A:** Translate-Key.req **S:** Wait-Translate.conf					
Send-Key.conf					**P:** Result = 0 **A:** Pick-SA-Attrs.req **S:** Wait-Attrs else **A:** Create-SA.conf (result) **S:** Idle				
Make-Key.conf						**P:** Result = 0 **A:** Pick-SA-Attrs.req **S:** Wait-Attrs else **A:** Create-SA.conf (result) **S:** Idle			
Select.conf							**P:** Result = 0 **A:** Pick-SA-Attrs.req **S:** Wait-Attrs else **A:** Create-SA.conf (result) **S:** Idle		

Table 3-24—Create-SA REQUESTER (continued)

Event	**State**								
	Idle	**Wait-KM-Alg.rsp**	**Wait-Request.conf**	**Wait-Translate.conf**	**Wait-Send.conf**	**Wait-Make.conf**	**Wait-Select.conf**	**Wait-Attrs**	**Wait-Send.ind**
Pick-SA-Attrs.conf								**A:** Create-SA.conf (result); Commit.req **S:** Idle	
Send-key.ind									**A:** Send-Key.rsp; Pick-SA-Attrs.req **S:** Wait-Attrs

Table 3-25—Create-SA RESPONDER

Event	State									
	Idle	Wait-Select	Wait-Send	Wait-Ask-Key	Wait-Make	Wait-Request	Wait-Pick-Attrs	Wait-Translate	Wait-Create-rsp	Wait-commit
Pick-KM-Alg.ind	**P:** technique = 0 **A:** Pick-KM-Alg.rsp **S:** Wait-Select else **P:** technique = 1 **A:** Pick-KM-Alg.rsp **S:** Wait-Send else **P:** technique = 2 **A:** Pick-KM-Alg.rsp **S:** Wait-Ask-Key else **A:** Pick-KM-Alg.rsp **S:** Wait-Make									
Select-Key.ind	**A:** Select-Key.rsp (result) **S:** Wait-Pick-Attrs	**A:** Select-Key.rsp (result) **S:** Wait-Pick-Attrs								
Send-Key.ind	**A:** Send-Key.rsp (result) **S:** Wait-Pick-Attrs		**A:** Send-Key.rsp (result) **S:** Wait-Pick-Attrs							

Table 3-25—Create-SA RESPONDER (continued)

Event	State									
	Idle	**Wait-Select**	**Wait-Send**	**Wait-Ask-Key**	**Wait-Make**	**Wait-Request**	**Wait-Pick-Attrs**	**Wait-Translate**	**Wait-Create-rsp**	**Wait-commit**
Please-Send-Key.ind	**P:** KEKIdentifier = ktc **A:** Translate-Key.req **S:** Wait-Translate else **P:** KEKIdentifier = kdc **A:** Request-Key.req **S:** Wait-Request else **A:** Send-Key.req **S:** Wait-Send			**P:** KEKIdentifier = ktc **A:** Translate-Key.req **S:** Wait-Translate else **P:** KEKIdentifier = kdc **A:** Request-Key.req **S:** Wait-Request else **A:** Send-Key.req **S:** Wait-Send						
Make-Key.ind	**A:** Make-Key.rsp (result) **S:** Wait-Pick-Attrs				**A:** Make-Key.rsp (result) **S:** Wait-Pick-Attrs					
Request-Key.conf						**P:** Package returned? **A:** Send-Key.req **S:** Wait-Send else **A:** Translate-Key.req **S:** Wait-Translate				
Send-Key.conf			**S:** Wait-Pick-Attrs							
Pick-Attr.ind							**A:** Create-SA.ind **S:** Wait-Create-rsp			

Table 3-25—Create-SA RESPONDER (continued)

Event	**State**									
	Idle	**Wait-Select**	**Wait-Send**	**Wait-Ask-Key**	**Wait-Make**	**Wait-Request**	**Wait-Pick-Attrs**	**Wait-Translate**	**Wait-Create-rsp**	**Wait-commit**
Translate-key.conf								**P:** Package returned? **A:** Send-Key.req **S:** Wait-Send else **A:** Translate-Key.req **S:** Wait-Translate		
Create-SA.rsp									**A:** Pick-SA-Attrs.rsp **S:** Wait-Commit	
Commit. ind										**S:** Idle

Table 3-26—Spawn-SA REQUESTER

Event	State					
	Idle	**Wait-Same-Key**	**Wait-Protected-Make-Key**	**Wait-Update-Key**	**Wait-Send-Key**	**Wait-Attrs**
Spawn-SA.req	**P:** Spawn-Option = 0 **A:** Spawn-Key.req (Identity transformation) **S:** Wait-Same-Key else **P:** Spawn-Option = 1 **A:** Spawn-Key.req **S:** Wait-Update-Key else **P:** Spawn-Option = 2 **A:** Protected-Make-Key.req **S:** Wait-Protected-Make-key else **A:** Send-Key.req **S:** Wait-Send-Key					
Spawn-Key.conf		**P:** Result = 0 **A:** Pick-SA-Attrs.req **S:** Wait-Attrs else **A:** Spawn-SA (result) **S:** Idle		**P:** Result = 0 **A:** Pick-SA-Attrs.req **S:** Wait-Attrs else **A:** Spawn-SA.conf (result) **S:** Idle		
Protected-Make-Key.conf			**P:** Result = 0 **A:** Pick-SA-Attrs.req **S:** Wait-Attrs else **A:** Spawn-SA.conf (result) **S:** Idle			
Send-Key.conf					**P:** Result = 0 **A:** Pick-SA-Attrs.req **S:** Wait-Attrs else **A:** Spawn-SA.conf (result) **S:** Idle	
Pick-SA-Attrs.conf						**A:** Spawn-SA.conf (result); Commit.req **S:** Idle

Table 3-27—Spawn-SA RESPONDER

Event	State			
	Idle	**Wait-Pick-Attrs.ind**	**Wait-Choose-Attrs**	**Wait-Commit**
Spawn-Key.ind	**A:** Spawn-Key.rsp **S:** Wait-Pick-Attrs.ind			
Pick-SA-Attrs.ind		**A:** Spawn-SA.ind **S:** Wait-Choose-Attrs		
Spawn-SA.rsp			**A:** Pick-SA-Attrs.rsp **S:** Wait-Commit	
Protected-Make-Key.ind	**A:** Protected-Make-Key.rsp **S:** Wait-Pick-Attrs.ind			
Send-Key.ind	**A:** Send-Key.rsp **S:** Wait-Pick-Attrs.ind			
Commit.ind				**S:** Idle

Table 3-28—Create-MSA REQUESTER

Event	State								
	Idle	**Wait-KM-Alg.rsp**	**Wait-Request.conf**	**Wait-Translate.conf**	**Wait-Send.conf**	**Wait-Make.conf**	**Wait-Select.conf**	**Wait-Attrs**	**Wait-Send.ind**
Create-MSA.req	**P:** (KMTL) > 1[a] or “must announce security policy” **A:** Pick-KM-Alg.req **S:** Wait-KM-Alg.rsp else **P:** technique = 0 **A:** Select-Key.req **S:** Wait-select.conf else **P:** technique = 3 **A:** Make-Key.req **S:** Wait-Make.conf else **P:** technique = 2 **A:** Please-Send-Key.req **S:** Wait-Send.ind else **P:** Local Key Gen allowed =Yes **A:** Translate-Key.req **S:** Wait-Translate.conf else **A:** Request-Key.req **S:** Wait-Request.conf								

[a] Key Management Technique List contains more than one entry.

Table 3-28—Create-MSA REQUESTER (continued)

Event	State								
	Idle	**Wait-KM-Alg.rsp**	**Wait-Request.conf**	**Wait-Translate.conf**	**Wait-Send.conf**	**Wait-Make.conf**	**Wait-Select.conf**	**Wait-Attrs**	**Wait-Send.ind**
Pick-KM-Alg.conf		**P:** technique = 0 **A:** Select-Key.req **S:** Wait-select.conf else **P:** technique = 3 **A:** Make-Key.req **S:** Wait-Make.conf else **P:** technique = 2 **A:** Please-Send-Key.req **S:** Wait-Send.ind else **P:** Local Key Gen allowed = Yes **A:** Translate-Key.req **S:** Wait-Translate.conf else **A:** Request-Key.req **S:** Wait-Request.conf							
Request-Key.conf			**P:** Package is returned? **A:** Send-Key.req **S:** Wait-Send.conf else **A:** Translate-Key.req **S:** Wait-Translate.conf						

Table 3-28—Create-MSA REQUESTER (continued)

Event	**State**								
	Idle	**Wait-KM-Alg.rsp**	**Wait-Request.conf**	**Wait-Translate.conf**	**Wait-Send.conf**	**Wait-Make.conf**	**Wait-Select.conf**	**Wait-Attrs**	**Wait-Send.ind**
Translate-Key.conf				**P:** Package is returned? **A:** Send-Key.req **S:** Wait-Send.conf else **A:** Translate-Key.req **S:** Wait-Translate.conf					
Send-Key.conf					**P:** Result = 0 **A:** Get-Mkey.req **S:** Wait-Mkey else **A:** Create-MSA.conf (result) **S:** Idle				
Make-Key.conf						**P:** Result = 0 **A:** Get-Mkey.req **S:** Wait-MKey else **A:** Create-MSA.conf (result) **S:** Idle			
Select.conf							**P:** Result = 0 **A:** Get-Mkey.req **S:** Wait-MKey else **A:** Create-MSA.conf (result) **S:** Idle		

Table 3-28—Create-MSA REQUESTER (continued)

Event	State								
	Idle	**Wait-KM-Alg.rsp**	**Wait-Request.conf**	**Wait-Translate.conf**	**Wait-Send.conf**	**Wait-Make.conf**	**Wait-Select.conf**	**Wait-Attrs**	**Wait-Send.ind**
Get-Mkey.conf								**A:** Create-MSA.conf (result) **S:** Idle	
Send-Key.ind									**A:** Send-Key.resp; Get-Mkey.req **S:** Wait-Mkey

Table 3-29—Create-MSA RESPONDER

Event	State								
	Idle	Wait-Select	Wait-Send	Wait-Ask-Key	Wait-Make	Wait-Request	Wait-Translate	Wait-Create	Wait-Get-Mkey
Pick-KM-Alg.ind	**P:** technique = 0 **A:** Pick-KM-Alg.rsp **S:** Wait-Select else **P:** technique = 1 **A:** Pick-KM-Alg.rsp **S:** Wait-Send else **P:** technique = 2 **A:** Pick-KM-Alg.rsp **S:** Wait-Ask-Key else **A:** Pick-KM-Alg.rsp **S:** Wait-Make								
Select-Key.ind	**A:** Select-Key.rsp (result) **S:** Wait-Get-Mkey	**A:** Select-Key.rsp (result) **S:** Wait-Get-Mkey							
Send-Key.ind	**A:** Send-Key.rsp (result) **S:** Wait-Get-Mkey		**A:** Send-Key.rsp (result) **S:** Wait-Get-Mkey						
Please-Send-Key.ind	**P:** KEKIdentifier = ktc **A:** Translate-Key.req **S:** Wait-Translate else **P:** KEKIdentifier = kdc **A:** Request-Key.req **S:** Wait-Request else **A:** Send-Key.req **S:** Wait-Send			**P:** KEKIdentifier = ktc **A:** Translate-Key.req **S:** Wait-Translate else **P:** KEKIdentifier = kdc **A:** Request-Key.req **S:** Wait-Request else **A:** Send-Key.req **S:** Wait-Send					

Table 3-29—Create-MSA RESPONDER (continued)

Event	State								
	Idle	**Wait-Select**	**Wait-Send**	**Wait-Ask-Key**	**Wait-Make**	**Wait-Request**	**Wait-Translate**	**Wait-Create**	**Wait-Get-Mkey**
Make-Key.ind	**A:** Make-Key.rsp (result) **S:** Wait-Get-Mkey				**A:** Make-Key.rsp (result) **S:** Wait-Get-Mkey				
Request-Key.con						**P:** Package returned? **A:** Send-Key.req **S:** Wait-Send else **A:** Translate-Key.req **S:** Wait-Translate			
Send-Key.conf			**S:** Wait-Get-Mkey						
Create-MSA.rsp								**A:** Get-Mkey.rsp **S:** Idle	
Translate-Key.conf							**P:** Package returned? **A:** Send-Key.req **S:** Wait-Send else **A:** Translate-Key.req **S:** Wait-Translate		
Get-Mkey.ind									**A:** Create-MSA.ind **S:** Wait-Create

Table 3-30—Spawn-MSA REQUESTER

Event	State	
	Idle	Wait-Next-MKey.conf
Spawn-MSA.req	**A:** Get-Next-MKey.req **S:** Wait-Next-MKey.conf	—
Get-Next-MKey.conf	—	**A:** Spawn-MSA.conf **S:** Idle

Table 3-31—Spawn-MSA RESPONDER

Event	State	
	Idle	Wait-Spawn-MSA.rsp
Get-Next-Mkey.ind	**A:** Spawn-MSA.ind **S:** Wait-Spawn-MSA.rsp	—
Spawn-MSA.rsp	—	**A:** Get-Next-Mkey.rsp **S:** Idle

Table 3-32—Delete-SA REQUESTER

Event	State
	Idle
Delete-SA.req	**A:** Delete-Key.req **S:** Idle

Table 3-33—Delete-SA RESPONDER

Event	State
	Idle
Delete-Key.ind	**A:** Delete-SA.ind **S:** Idle

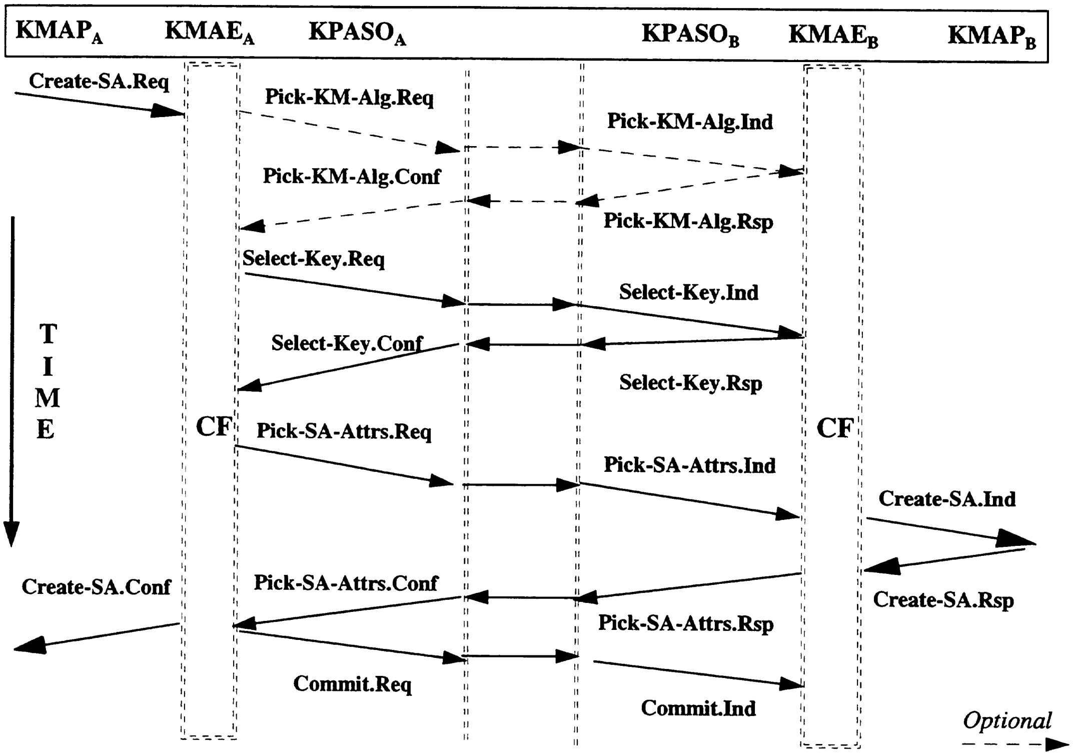
Table 3-34—Key Center RESPONDER

Event	State
	Idle
Request-Key.ind	**A:** Request-Key.rsp **S:** Idle
Translate-Key.ind	**A:** Translate-Key.rsp **S:** Idle

3.6.2 Sample timing diagrams

The following diagrams illustrate several paths through the state tables. These timing diagrams make no attempt to illustrate all possible paths through the state tables; rather, they illustrate paths that are expected to be used frequently. None of these timing diagrams illustrate error cases.

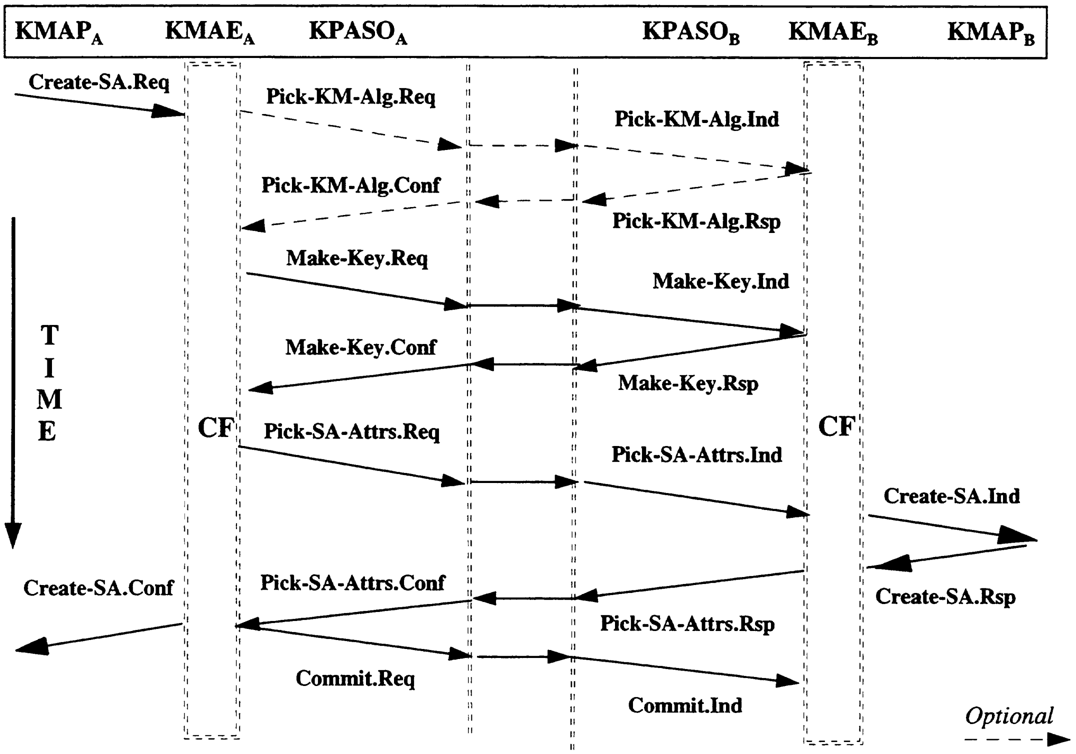
Create-SA for Manual Based



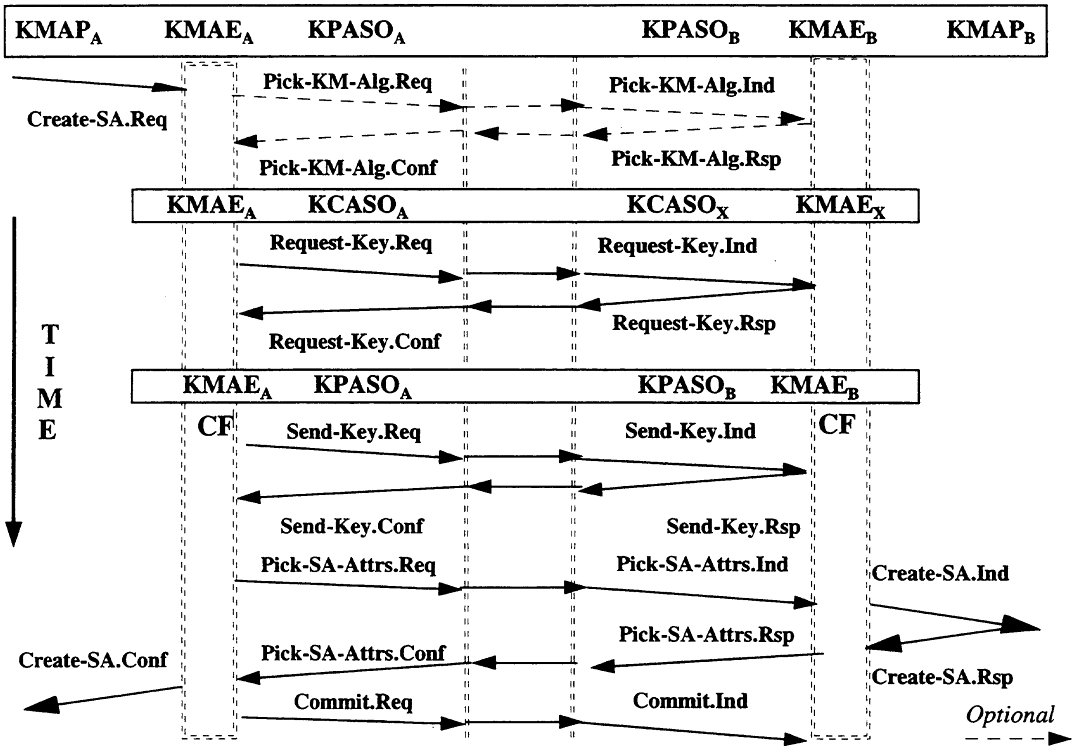
NOTE

In these timing diagrams,
subscript A = initiator;
subscript B = responder;
subscript X = KDC or KTC.

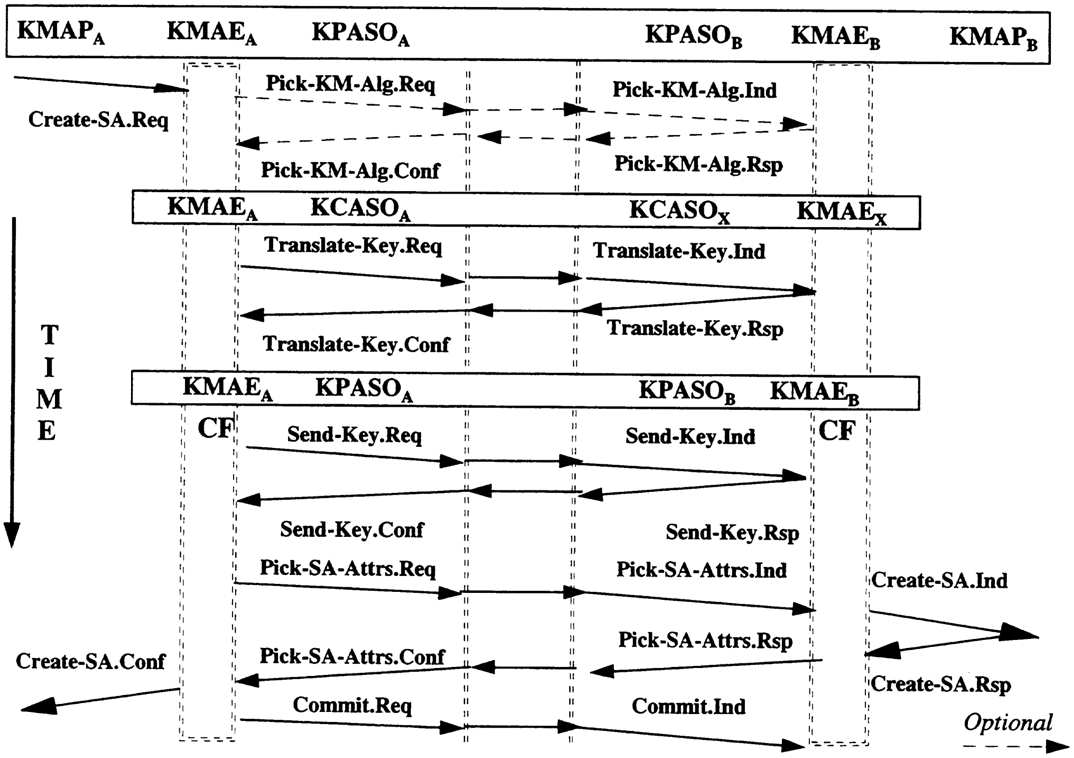
Create-SA for Certificate Based



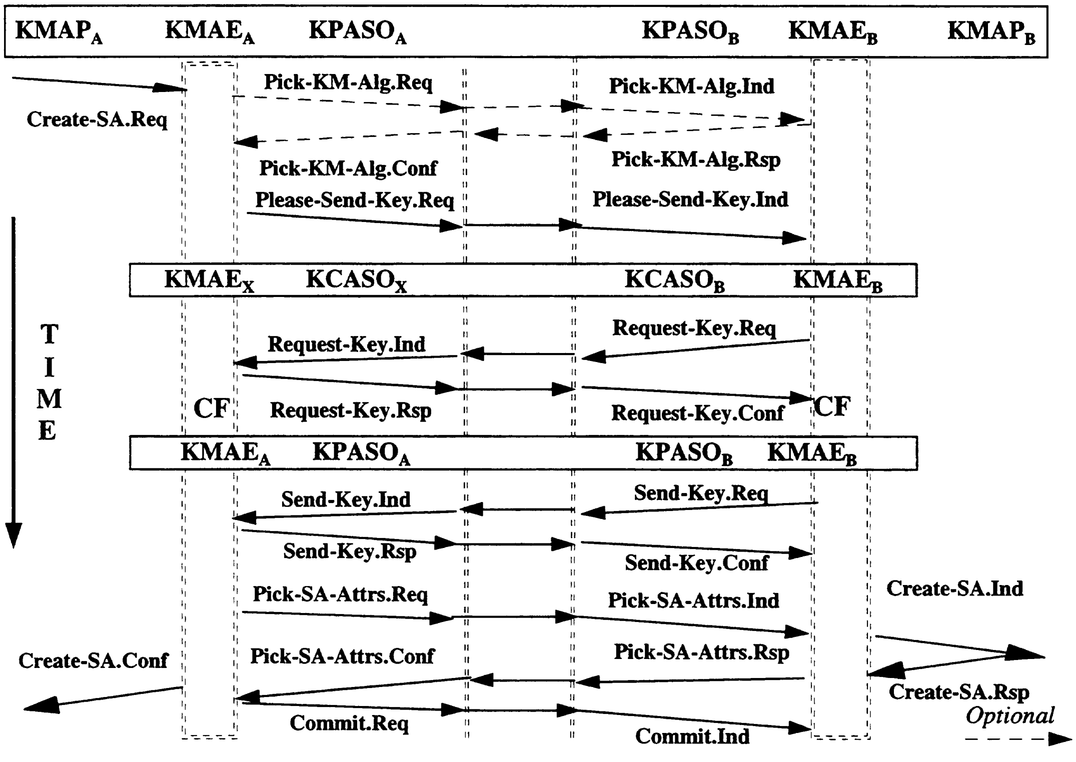
Create-SA for Center Based via KDC



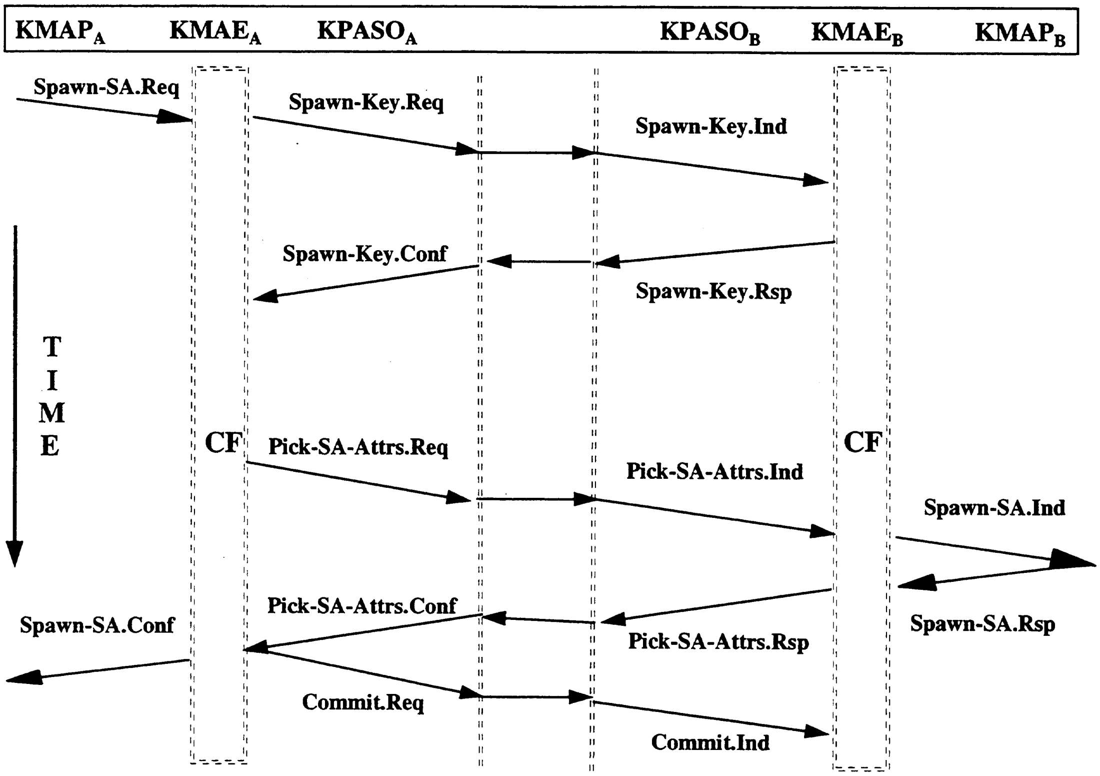
Create-SA for Center Based via KTC



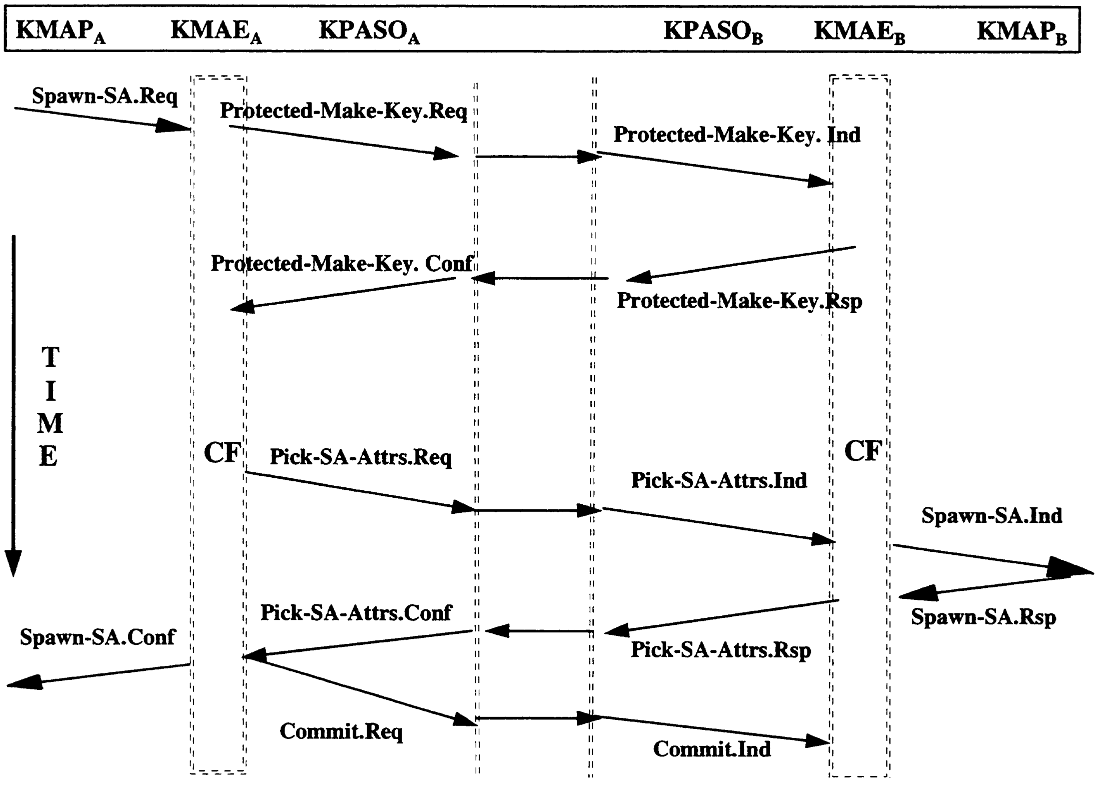
Create-SA for Center Based via KDC$_B$



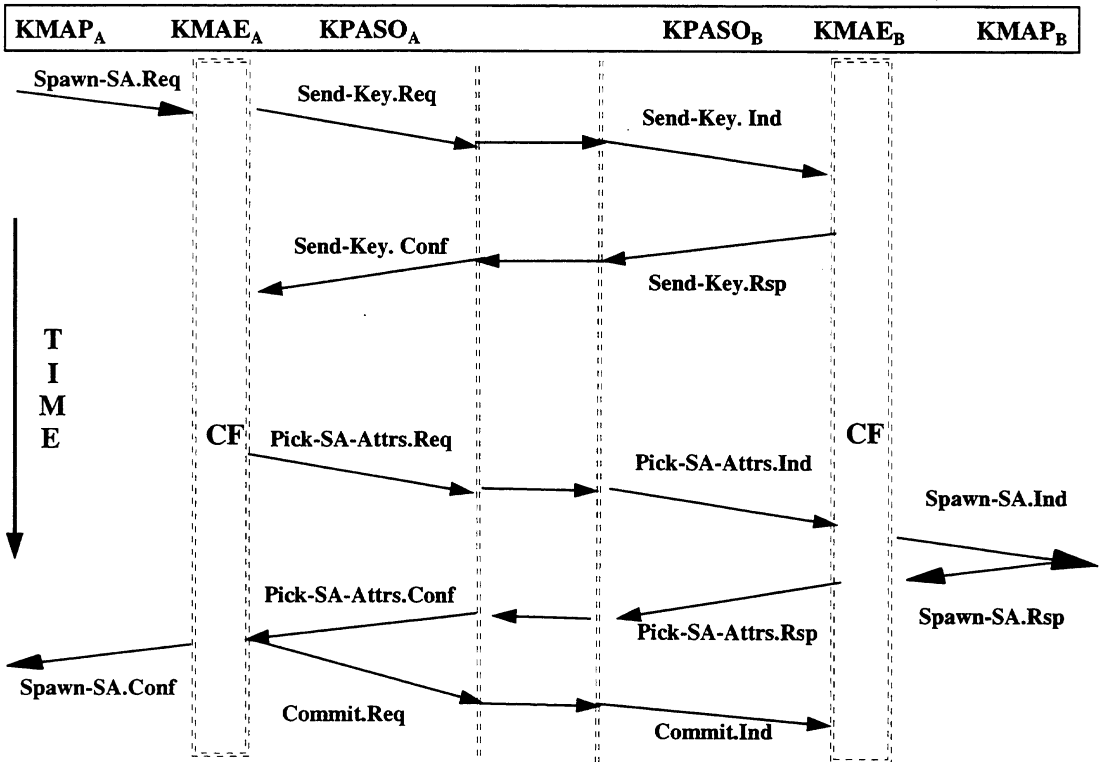
Spawn-SA for Key Transformation



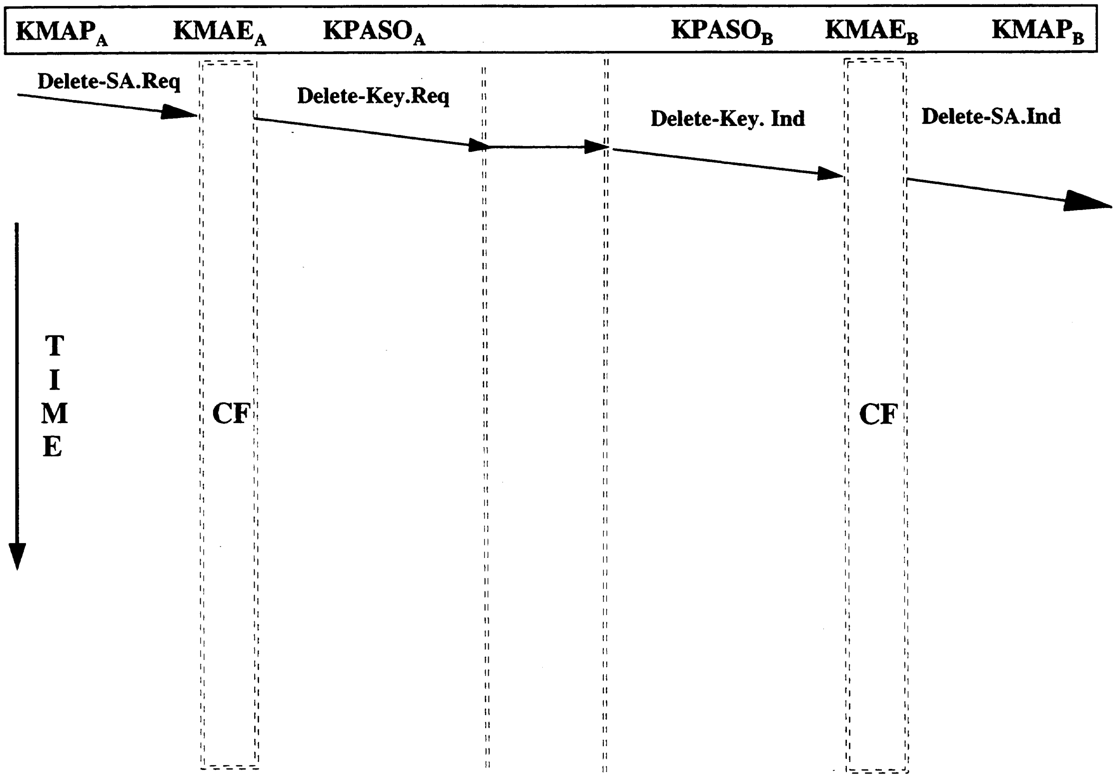
Spawn-SA for ProtectedCertificate Based



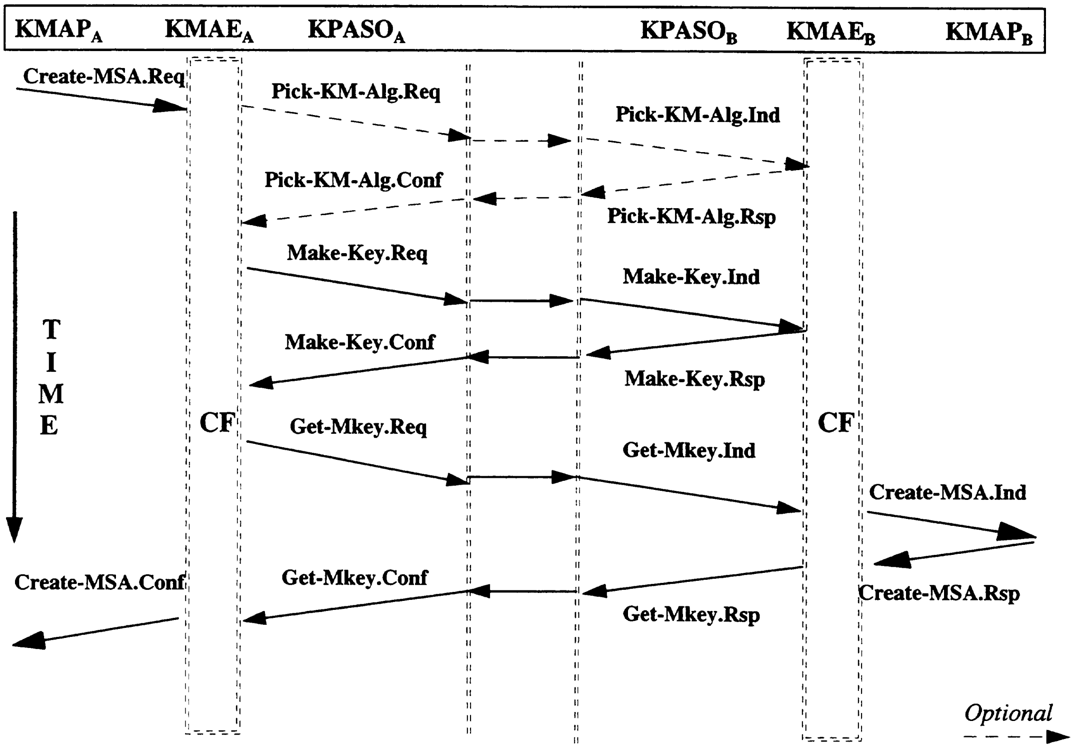
Spawn-SA for Center Based



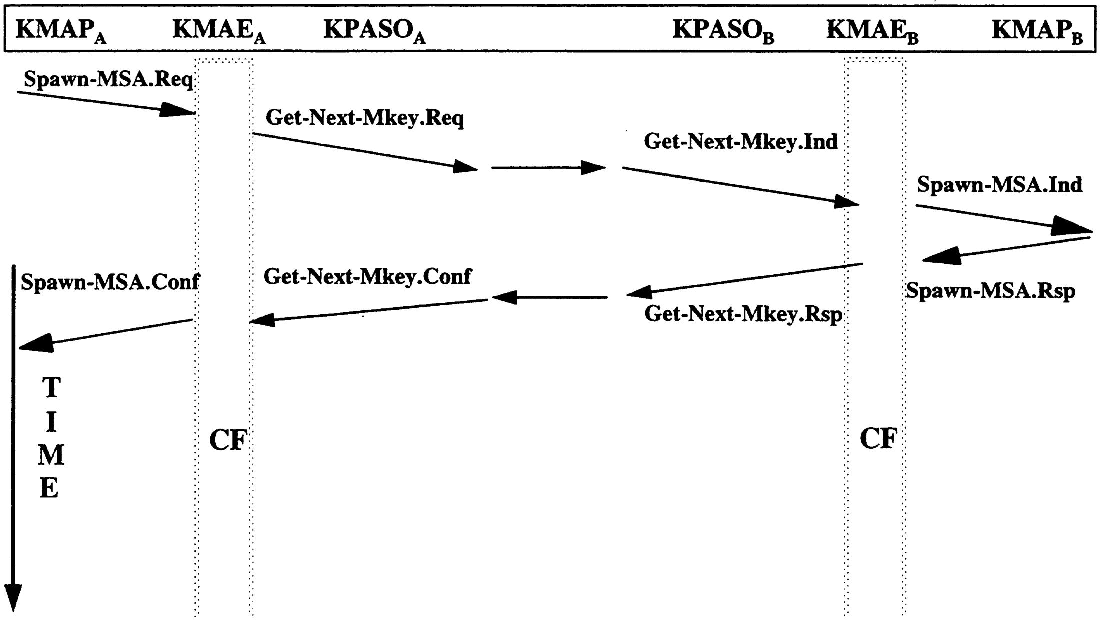
Delete-SA



Create-MSA for Certificate Based



Spawn-MSA



Add Annex 3A as follows:

Annex 3A

(normative)

Locating SDE key management entities

When the Secure Data Exchange (SDE) protocol is implemented in bridges, the internetwork is divided into two logical internetworks. The traffic from stations on the internetwork behind SDE bridges is protected (confidentiality, integrity, or both) when it is carried across the other internetwork.

The bridges with connections to each of the two internetworks form separate spanning trees as defined in IEEE Std 802.1D-1990. The SDE bridges have ports in both spanning trees. The internetwork that carries SDE protected traffic appears as one logical LAN to the other internetwork.

When an unprotected PDU arrives at an SDE bridge, the source and destination addresses reveal the stations involved in the communication, but neither address reveals the address of the peer SDE bridge. This leads to an address discovery problem. To establish a security association in support of these SDE entities, their addresses must be discovered. Probe frames are used by the SDE bridges to locate the correct SDE bridge. Once the addresses are known, the KMAPs within the SDE bridges communicate to establish the security association.

Whenever the spanning tree changes, the SDE bridges must verify that the peer SDE bridge is still the active path. It is possible that the altered spanning tree blocks the path between the two communicating SDE bridges. Should the path between the two SDE bridges become blocked, the address discovery process must be repeated to locate the SDE bridge that has taken over responsibility, and then the KMAP must establish a new security association on behalf of that SDE entity.

3A.1 Probe frames

When SDE is implemented in bridges, the local KMAP must locate the remote KMAP before the application-association for KMP can be established. The local KMAP sends a probe request frame addressed to the destination MAC address. The probe request frame will be handled by any bridges necessary to deliver the frame; however, an SDE-enabled bridge will intercept the probe request frame and return a probe response frame. Note that SDE-enabled end systems must be prepared to process probe request and probe response frames; otherwise, the address resolution will fail.

Figure 3A.1 shows the format of the probe request frame and the probe response frame. For simplicity, frames have the same format.

SDE Designator	SAID	Type	Source	Dest	Source KM	Dest KM
3	4	1	8	8	8	8

OCTETS

Figure 3A.1—Probe frame format

3A.1.1 Probe request

The local KMAP generates a probe request frame as follows:

- SDE Designator is the reserved LSAP for SDE, which is “0A0A03.” Use of this LSAP ensures that SDE-enabled bridges will intercept this frame.
- SAID is a reserved SDE SAID. It is all zeros. Use of this SAID value ensures that the receiving SDE entity will process this frame as management traffic, not user traffic.
- Type indicates that this frame is a probe request. The value is 1.
- Source is the source MAC address for the traffic that will be protected by SDE once the security association is established.
- Dest is the destination MAC address for the traffic that will be protected by SDE once the security association is established.
- Source KM is the source MAC address for the KMAP supporting the SDE entity that will handle traffic on behalf of the source MAC address. If the SDE entity is located in the end system, then Source and Source KM have the same MAC address.
- Dest KM is unknown; the field must contain all zeros.

The local KMAP sends the probe request in a MAC frame addressed to the destination MAC address.

3A.1.2 Probe processing

When the KMAP receives a probe request, the processing depends on the location of the KMAP.

If the KMAP is located at the end system, then it simply fills the Dest KM field with its MAC address, and sends the probe response frame back to the KMAP that originated the probe request.

If the KMAP is located at an SDE-enabled bridge, then it must determine if the destination MAC address is reachable on the plaintext side of the SDE-enabled bridge. If the SDE-enabled bridge is configured with a list of MAC addresses that it supports, then a simple list lookup can determine whether or not the destination MAC address is supported. Learning from previous traffic processed by the SDE-enabled bridge may determine whether or not the destination MAC address is supported. Otherwise, the SDE-enabled bridge must send an LLC test request frame (as specified in ISO/IEC 8802-2: 1994 Section 5.4.1.1.3) to the destination MAC address to determine if that MAC address is reachable on the plaintext side of the SDE-enabled bridge. If the SDE-enabled bridge has multiple unblocked plaintext ports, the test frames must be sent on each one. If the LLC test response frame (as specified in ISO/IEC 8802-2: 1994 Section 5.4.1.2.2) is returned, then the KMAP fills the Dest KM field with its MAC address, and sends the probe response frame back to the KMAP that originated the probe request. If no test response frame is returned, then no action is taken.

3A.1.3 Probe response

The local KMAP generates a probe response frame as follows:

- SDE Designator is the reserved LSAP for SDE, which is “0A0A03.”
- SAID is a reserved SDE SAID. It is all zeros.
- Type indicates that this frame is a probe response. The value is 2.
- Source is the source MAC address for the traffic that will be protected by SDE once the security association is established. This value is copied from the probe request frame.
- Dest is the destination MAC address for the traffic that will be protected by SDE once the security association is established. This value is copied from the probe request frame.

- Source KM is the MAC address for the KMAP that sent the probe request. This value is copied from the probe request frame.
- Dest KM is the MAC address for the responding KMAP.

The KMAP sends the probe response in a MAC frame addressed to the MAC address of the KMAP that sent the probe request.

3A.2 Address discovery scenario

This clause describes an example of the address discovery process. Figure 3A.2 is used in this text to describe a scenario of the operation of SDE equipped bridges. In the diagram, Bridges X, Y, and Z have SDE implemented.

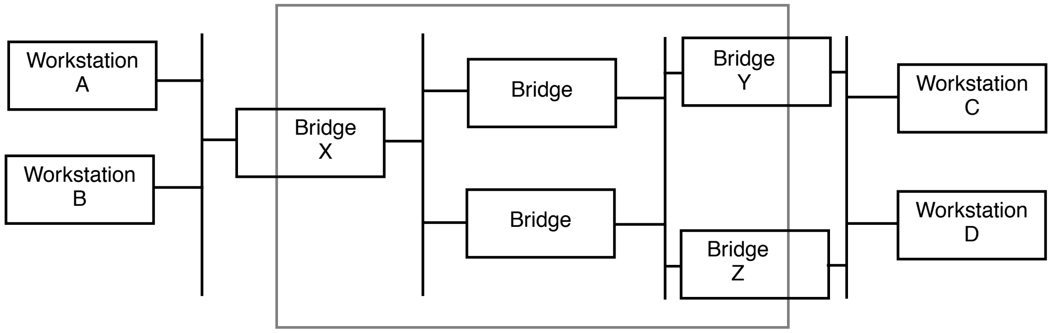


Figure 3A.2—Address discovery example

a) Workstation A knows Workstation D’s MAC layer address, but Workstation D has never transmitted any link layer frames and none of the bridges know of Workstation D.

b) Workstation A sends a frame with source address = A and destination address = D.

c) Bridge X does not have a Security Association for source address = A and destination address = D. Bridge X does not have an entry in its flow tables for destination address = D. Bridge X floods an SDE Probe packet on all active ports, except the port on which Bridge X received the SDE Probe packet.

d) Bridges Y and Z receive the Probe packet. Neither has information for destination address = D. Both Bridges Y and Z flood Test frames on all active ports, except the port on which Bridges Y and Z received the SDE Probe packet.

e) Workstation D responds to the test frame from Bridge Z.

f) Bridge Z responds to the Probe from Bridge X.

Add Annex 3B as follows:

Annex 3B

(informative)

Certificate replacement

3B.1 Service definition (Cert-Replace)

The Cert-Replace is a KMAE service that provides the capability for a peer key management system to request a replacement X.509 certificate. In addition, the replacement certificate could be received with optional replacement of the associated private material from a responding peer key management system. The decision to replace the private material is determined by the responding peer key management system. Cert-Replace is a confirmed service that establishes the replacement of certificates, which are stored in the SMIB of the requesting entity. The certificate that is being replaced can be passed in or obtained from the SMIB.

This service requires a security-association with the certification authority to exist prior to invocation.

The parameters of Cert-Replace are defined in Table 3B.1.

Table 3B.1—Cert-Replace parameters

Cert-Replace parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M (=)	M (=)	3.4.1.1.2
Certificate-To-Be-Replaced	U	C (=)	—	—	3B.1.1
Replacement-Indicator	—	—	M	M (=)	3B.1.2
Calling SAID	—	—	M	M (=)	3.4.1.1.7
Called SAID	M	M (=)	—	—	3.4.1.1.8
Result	—	—	M	M (=)	3.4.1.1.9

3B.1.1 Certificate-To-Be-Replaced

Certificate-To-Be-Replaced represents the X.509 certificate that is being replaced.

```
Certificate-To-Be-Replaced::= Certificate
```

3B.1.2 Replacement-Indicator

Replacement Material is a Boolean value that indicates if the private keying material associated with the certificate has been replaced.

```
Replacement-Indicator   ::= BOOLEAN
```

3B.2 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.

The parameters of Replace-Certificate are defined in Table 3B.2.

Table 3B.2—Replace-Certificate parameters

Replace-Certificate parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M (=)	M (=)	3.4.1.1.2
Certificate-To-Be-Replaced	M	M (=)	—	—	3B.1.1
Replacement-Material	—	—	M	M (=)	3B.2.1
Calling SAID	—	—	M	M (=)	3.4.1.1.7
Called SAID	M	M (=)	—	—	3.4.1.1.8
Result	—	—	M	M (=)	3.4.1.1.9

3B.2.1 Replacement-Material

Replacement-Material is the replacement X.509 certificate and optionally the associated private material.

```
Replacement-Material    ::=   SEQUENCE {
                  certificate       CertificationPath,
                  privateMaterial   OCTET STRING OPTIONAL}
```

3B.3 Security exchanges

```
certificate-replacement       SECURITY-EXCHANGE::=
{
      SE-ITEMS    {request-replacement, replacement-response}
      IDENTIFIER  { se-id-cert-replacement(14)}
}
request-replacement     SEC-EXCHG-ITEM::=
{
      ITEM TYPE   Replacement-rq
      ERRORS      {replacement-failed}
      ITEM ID1
}
replacement-response    SEC-EXCHG-ITEM::=
{
      ITEM TYPE   Replacement-rp
      ITEM ID2
}
```

```
replacement-failed      SE-ERROR ::=
{
      PARAMETER   Result
      ERROR CODE  14
}
Replacement-rq::=  SEQUENCE  {
      CalledSAID,
      PROTECTED {Certificate-To-Be-Replaced, enciphered}   }

Replacement-rp::=  SEQUENCE  {
      CallingSAID,
      PROTECTED  { Replacement-Material, enciphered }  }

Certificate-To-Be-Replaced::=Certificate

Replacement-Material::=SEQUENCE {
      certificate       CertificationPath,
      privateMaterial   OCTET STRING OPTIONAL}
```

Add Annex 3C as follows:

Annex 3C

(informative)

Compromised material lists

3C.1 Security definition (CML-Request)

The CML-Request is a KMAE service that provides the capability for a peer key management system to obtain a compromised material list. The CML is a method for notifying the peer key management system of compromised material.

This service could be used to distribute X.509 Certificate Revocation Lists. However, the purpose of this service is much broader, in that it could be used for distributing a list of identifiers of any keying material that has been compromised.

This service requires a security-association with the certification authority to exist prior to invocation.

The parameters of CML-Request are defined in Table 3C.1.

Table 3C.1—CML-Request parameters

CML-Request parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M (=)	M (=)	3.4.1.1.2
Name	M	M (=)	—	—	3C.1.1
Attribute-Id	M	M (=)	—	—	3C.1.2
Attribute-Value	—	—	M	M (=)	3C.1.3
Calling SAID	—	—	M	M (=)	3.4.1.1.7
Called SAID	M	M (=)	—	—	3.4.1.1.8
Result	—	—	M	M (=)	3.4.1.1.9

3C.1.1 Name

Name is the Distinguished Name of the issuer of the compromised material list.

```
Name              ::= DistinguishedName
```

3C.1.2 Attribute-Id

Attribute-Id is the object identifier of the compromised material list. It is used by the requester to indicate the type of the compromised material list that the requester desires.

```
Attribute-Id      ::= OBJECT IDENTIFIER
```

3C.1.3 Attribute-Value

The Attribute-Value is the compromised material list.

```
Attribute-Value   ::= OCTET STRING
```

3C.2 Request-CML

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

The parameters of Request-CML are defined in Table 3C.2.

Table 3C.2—Request-CML parameters

Request-CML parameter name	.req	.ind	.rsp	.conf	Reference
Calling AE-Title	M	M (=)	—	—	3.4.1.1.1
Called AE-Title	M	M (=)	M (=)	M (=)	3.4.1.1.2
Name	M	M (=)	—	—	3C.1.1
Attribute-Id	M	M (=)	—	—	3C.1.2
Attribute-Value	—	—	M	M (=)	3C.1.3
Calling SAID	—	—	M	M (=)	3.4.1.1.7
Called SAID	M	M (=)	—	—	3.4.1.1.8
Result	—	—	M	M (=)	3.4.1.1.9

3C.3 Security exchanges

```
request-CML       SECURITY-EXCHANGE::=
{
      SE-ITEMS    {cml-request, cml-response}
      IDENTIFIER  {se-id-request-cml(15)}
}
cml-request       SEC-EXCHG-ITEM::=
{
      ITEM TYPECML-rq
      ERRORS      {cml-failed}
      ITEM ID1
}
```

```
cml-response      SEC-EXCHG-ITEM::=
{
      ITEM TYPECML-rp
      ITEM ID2
}
cml-failed        SE-ERROR ::=
{
      PARAMETER   Result
      ERROR CODE  15
}
CML-rq      ::= SEQUENCE{
                        CalledSAID,
                        PROTECTED {
                              SEQUENCE {
                                    Name,
                                    Attribute-Id  }, enciphered }  }
CML-rp      ::= SEQUENCE {
                        CallingSAID,
                        PROTECTED {
                              SEQUENCE {
                                    Name,
                                    Attribute-Id,
                                    Attribute-Value  }, enciphered }  }

Name                    ::= DistinguishedName
Attribute-Id            ::= OBJECT IDENTIFIER
Attribute-Value         ::= OCTET STRING
```

Add Annex 3D as follows:

Annex 3D

(informative)

Key distribution scenarios

The scenarios in this annex illustrate the security association creation (Create-SA) for each of the three key management techniques supported by this protocol: manual key distribution, center-based key distribution, and certificate-based key distribution. Each scenario includes sample security exchanges to show which data is included in each of the security exchanges. Attribute negotiation is not included in these scenarios; however, Annex 3E illustrates attribute negotiation for the Secure Data Exchange protocol.

3D.1 Manual key distribution scenario

If the initiator wants to exchange protected traffic with the responder and more than one key management algorithm could be used, then the initiator sends to the responder the Pick-KM-Alg-rq to negotiate the key management algorithm. The responder responds with a Pick-KM-Alg-rp selecting the first key management algorithm offered by the initiator that is supported by the responder. In this scenario, manual key distribution is selected. The initiator sends a Select-Key-rq to the responder identifying predistributed keying material. The responder replies with a Select-Key-rp, confirming the initiator's choice of keying material. Once the keying material is selected, attributes are negotiated. The initiator sends a Pick-SA-Attrs-rq, and the responder replies with a Pick-SA-Attrs-rp.

A sample Select-Key-rq is shown below:

```
Select-Key-rq ::= SEQUENCE {
 CryptographicKeyingMaterialIdentifier"PTFKMC-3-9FD3AAD2F2691B9A",
 TransformAlgorithmIdentifier         -- absent --,
 AlgorithmIdentifier                  { 1 3 14 3 2 7 }, -- DES-CBC
 PROTECTED {  SEQUENCE {              -- encrypted using key identified
                                         above
  CallingSAID                         48,
  Nonce                               950704201545
 } }
}
```

A sample Select-Key-rp that matches the above Select-Key-rq is shown below:

```
Select-Key-rp ::= PROTECTED { SEQUENCE {
                                    --encrypted using key and algorithm
                                    --identified in the proceeding
                                      Select-Key-rq
  CallingSAID                       48,
  CalledSAID                        42,
  NoncePlus1                        950704201546
 } }
```

3D.2 Center-based key distribution scenario

If the initiator wants to exchange protected traffic with the responder and more than one key management algorithm could be used, then the initiator sends to the responder the Pick-KM-Alg-rq to negotiate the key management algorithm. The responder responds with a Pick-KM-Alg-rp selecting the first key management algorithm offered by the initiator that is supported by the responder. In this scenario, center-based key distribution is selected. Next, the initiator sends a Request-Key-rq to the Key Distribution Center (KDC) to request keying material for use by the initiator and responder. The KDC responds with a Request-Key-rp containing keying material encrypted in a key encryption key shared between the initiator and the KDC. The Request-Key-rp includes the same keying material encrypted in a key encryption key (KEK) shared between the responder and the KDC. The initiator sends to the responder a Send-Key-rq that contains the keying material that was encrypted for the responder. Then, the responder replies with a Send-Key-rp. Finally, attributes are negotiated. The initiator sends a Pick-SA-Attrs-rq, and the responder replies with a Pick-SA-Attrs-rp.

The format of the exchanges depends on the CENTER-PROTOCOL that is used. A sample CENTER-PROTOCOL registration is included for purposes of example. The OBJECT IDENTIFIER associated with these definitions is:

```
id-center ::= { iso(1) member-body(2) us(840) ieee-802dot10(10022) 22 1
}
center-based-exchanges CENTER-PROTOCOL {
      SEND-RQ-PARMS     { RequestRqParms }
      SEND-RS-PARMS     { RequestRsParms }
      CENTER-PROTOCOL-ID  {id-center}  }

RequestRqParms    ::=  SEQUENCE  {
                  AlgorithmIdentifier,
                  PROTECTED  {  SEQUENCE  {
                        Nonce,
                        CalledAETitle,
                        AlgorithmIdentifier
                  },  enciphered  }  }

RequestRsParms    ::=  PROTECTED  {  SEQUENCE  {
                        TitleOrPackage,
                        KeyingMaterial,
                        Nonce,
                        CalledAETitle,
                        AlgorithmIdentifier
                  },  enciphered  }

TranslateKeyParms  ::=  SEQUENCE  {
                  AlgorithmIdentifier,
                  PROTECTED  {  SEQUENCE  {
                        KeyingMaterial,
                        Nonce,
                        CalledAETitle,
                        AlgorithmIdentifier
                  },  enciphered  }  }

TranslateRsParms  ::=  RequestRsParms
```

```
SendRqParms       ::=  SEQUENCE  {
                  AlgorithmIdentifier,
                  BPackage,
                  PROTECTED  {  SEQUENCE  {
                        CallingSAID,
                        Time
                  },  enciphered  }  }

SendRsParms       ::=  PROTECTED  {  SEQUENCE  {
                        CallingSAID,
                        CalledSAID,
                        TimePlus1
                  },  enciphered  }

TitleOrPackage    ::=  CHOICE  {
      bPackage    [0] BPackage,
      title       [1] KTCTitle  }

BPackage          ::=  PROTECTED  {  SEQUENCE  {
                        KeyingMaterial,
                        AlgorithmIdentifier,
                        CallingAETitle
                  },  enciphered  }

KeyingMaterial    ::=  OCTET STRING
Nonce       ::=  INTEGER
Time        ::=  GeneralizedTime
TimePlus1   ::=  GeneralizedTime
```

A sample Request-Key-rq is shown below:

```
Request-Key-rq ::=  SEQUENCE  {
  callingTitle          (AETitle of the initiator KMAE),
  centerProtocol        {1 2 840 10022 22 1},
  SEQUENCE  {
    AlgorithmIdentifier { 1 3 14 3 2 7 },  -- DES-CBC
                        -- algorithm used to encrypt the following
                        -- PROTECTED SEQUENCE

    PROTECTED  {  SEQUENCE  {
      Nonce             960214201545,
      CalledAETitle     (AETitle of the responding KMAE),
      AlgorithmIdentifier{ 1 3 14 3 2 7 }, -- DES-CBC
                        -- the KDC will generate keying material
                        -- for use with DES-CBC
    }
 }
}
```

A sample Request-Key-rp that matches the above Request-Key-rq is shown below:

```
Request-Key-rp ::= SEQUENCE {
 kdcTitle               (AETitle of the KDC),
 PROTECTED {  SEQUENCE {-- encrypted in the key shared between the
                        -- initiator KMAE and the KDC
                        [0] -- a Bpackage follows

 PROTECTED { SEQUENCE {-- the initiator does not have the key to decrypt
                       -- this portion; it is for the responding KMAE
      KeyingMaterial    0123456789ABCDEF,
      AlgorithmIdentifier{ 1 3 14 3 2 7 }, -- DES-CBC
      CallingAETitle    (AETitle of the initiator KMAE),
    } }
    KeyingMaterial      0123456789ABCDEF,
    Nonce               960214201545,
    CalledAETitle       (AETitle of the responding KMAE),
    AlgorithmIdentifier { 1 3 14 3 2 7 },  -- DES-CBC
  }  }
}
```

A sample Send-Key-rp that matches the above Request-Key-rp is shown below:

```
Send-Key-rq ::=  SEQUENCE  {
  kekIdentifier         [1] (AETitle of the KDC),
  centerProtocol        {1 2 840 10022 22 1},
  SEQUENCE  {
    AlgorithmIdentifier,{ 1 3 14 3 2 7 },  -- DES-CBC
    PROTECTED  {  SEQUENCE  {-- the BPackage
      KeyingMaterial    0123456789ABCDEF,
      AlgorithmIdentifier{ 1 3 14 3 2 7 },  -- DES-CBC
      CallingAETitle    (AETitle of the initiator KMAE),
    }  }
    PROTECTED  {  SEQUENCE  {-- encrypted in the key inside the BPackage
      CallingSAID       48,
      Time              "19960214201548Z",
    }  }
  }
}
```

A sample Send-Key-rp that matches the above Send-Key-rq is shown below:

```
Send-Key-rp ::=  PROTECTED  {  SEQUENCE  {
  CallingSAID           48,
  CalledSAID            42,
  TimePlus1             "19960214201549Z"
}  }
```

3D.3 Certificate-based key distribution scenario

Two examples of certificate-based key distribution are included here: one using RSA key transport as defined in X9.44, and the other one using the X9.42 variation of Diffie-Hellman Key Agreement. Two examples are shown since the same security exchanges are used in different ways.

In either of the two examples, if the initiator wants to exchange protected traffic with another entity (the responder), and if more than one key management algorithm could be used, then the initiator sends to the responder the Pick-KM-Alg.req to negotiate the key management algorithm. The responder returns a Pick-KM-Alg.rsp selecting the first key management algorithm offered by the initiator that is supported by the responder. In both sample scenarios, certificate-based key distribution is selected.

3D.3.1 X9.44 RSA key transfer scenario

The initiator sends to the responder a Make-Key.req that includes the initiator's certificate. The initiator's certificate contains the initiator's RSA public key used for key management. The responder replies with a Make-Key.rsp that includes the responder's certificate, symmetric keying material encrypted in the initiator's RSA public key, and a signature over the encrypted keying material. The responder's certificate contains the responder's RSA public key used for digital signatures. Encrypting the symmetric keying material in the initiator's RSA public key ensures that only the initiator can decrypt the keying material, and the signature authenticates the responder as the source of the keying material. Once the keying material is transferred, attributes are negotiated. The initiator sends a Pick-SA-Attrs.req, and the responder replies with a Pick-SA-Attrs.rsp.

The format of the exchanges depends on the KEY-GEN-ALG that is used. The OBJECT IDENTIFIER associated with X9.44 RSA Key Transport is:

```
{ iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 22 }
The definitions for Rq-Parms and Rp-Parms for X9.44 RSA Key Transport are:

Rq-Parms  ::=  NULL
Rp-Parms  ::=  X944KeyMaterial
X944KeyMaterial  ::=  PROTECTED  {  EncipheredKey, signed  }
EncipheredKey  ::=  OCTET STRING
```

A sample Make-Key.req is shown below:

```
MakeKeyRq  ::=  SEQUENCE  {
  keyGenAlgorithmID{ 1 3 14 3 2 22 },
  CallingCertPath       (Initiator certification path; contains the
                         initiator's key management RSA public key.)
  Rq-Params             NULL
  CallingSAID           48
}
```

A sample Make-Key.rsp that matches the above Make-Key.req is shown below:

```
MakeKeyRs  ::=  SEQUENCE  {
  CalledCertPath        (Responder certification path; contains the
                        responder's signature RSA public key.)
  PROTECTED  {          --  signed with the responder's RSA private key
    EncipheredKey       (Symmetric key encrypted in the initiator's
                          RSA public key.)
  }
  CallingSAID           48,
  CalledSAID            42
}
```

3D.3.2 X9.42 Diffie-Hellman key agreement scenario

The initiator sends to the responder a Make-Key.req that includes the initiator's certificate and a nonrepeating or random value created by the initiator, called Ra. The initiator's certificate contains the initiator's Diffie-Hellman public key. The responder replies with a Make-Key.rsp that includes the responder's certificate and a non-repeating or random value created by the responder, called Rb. From these values, the initiator and the responder can generate the same symmetric keying material. The initiator uses his own Diffie-Hellman private key, the responder's Diffie-Hellman public key from the responder's certificate, the public Diffie-Hellman generator and prime modulus, Ra, and Rb to generate the keying material. The responder generates the same keying material from his own Diffie-Hellman private key, the initiator's Diffie-Hellman public key from the initiator's certificate, the public Diffie-Hellman generator and prime modulus, Ra, and Rb. Details of the calculations can be found in X9.42. Once the keying material is generated, attributes are negotiated. The initiator sends a Pick-SA-Attrs.req, and the responder replies with a Pick-SA-Attrs.rsp.

The format of the exchanges depends on the KEY-GEN-ALG that is used.The OBJECT IDENTIFIER associated with X9.42 Diffie-Hellman Key Agreement is:

```
 { iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 25 }
```

The definitions for Rq-Parms and Rp-Parms for X9.42 Diffie-Hellman Key Agreement are:

```
Rq-Parms  ::=  X942Params
Rp-Parms  ::=  X942Params
X942Params  ::=  SEQUENCE  {
      ephemeralKeyDHPublicKey OPTIONAL,  -- not used in this scenario
      nrv               OCTET STRING OPTIONAL  }
```

A sample Make-Key.req is shown below:

```
MakeKeyRs  ::=  SEQUENCE  {
  keyGenAlgorithmID     { 1 3 14 3 2 25 },
  CallingCertPath       (Initiator certification path; contains the
                           initiator's Diffie-Hellman public key.)
  SEQUENCE  {
    nrv                 (Random value)
  }
  CallingSAID           48
}
```

A sample Make-Key.rsp that matches the above Make-Key.req is shown below:

```
MakeKeyRs  ::=  SEQUENCE  {
  CalledCertPath        (Responder certification path; contains the
                          responder's Diffie-Hellman public key.)
  SEQUENCE  {
    nrv                 (Random value)
  }
  CallingSAID           48,
  CalledSAID            42
}
```

Add Annex 3E as follows:

Annex 3E

(normative)

SDE attribute negotiations

This annex defines the attribute negotiation for SDE. The initiator provides a sealed and encrypted security protocol attribute list. Within each attribute list member, a list of confidentiality and integrity algorithms can be specified. The initiator determines the SDE attributes that are acceptable, and the initiator provides the alternatives in order of decreasing preference. The responder chooses the first alternative that is acceptable. The responder chooses one of the items from the attribute list and one of the confidentiality and integrity algorithm alternatives.

Should an implementor wish to augment the attributes assigned in this annex, that implementor shall assign a new OBJECT IDENTIFIER and register a new attribute structure. Implementors who register an augmented attribute structure are encouraged to also support the attribute structure specified in this annex to ensure interoperability with other implementations.

3E.1 Security attributes for Secure Data Exchange (SDE)

The initiator structures the list according to the ASN.1 defined for the particular security protocol. KMP provides the SPAttrs field to carry the initiator attribute alternatives.

```
SPAttrs ::= SEQUENCE {
 spAttrs      SP-ATTRS.&attrs-Id,
 attrs        SP-ATTRS.&Sp-Attrs { @spAttrs } OPTIONAL }
```

spAttrs is set to the OID for SDE. The SDE OID is:

```
  {iso(1) member-body(2) us(840) ieee-802dot10(10022) asn1Module(2) sde-
Attrs(2)}
```

The following ASN.1 structure is associated with the SDE OID:

```
SDEAttrs  ::=     SEQUENCE  {
  assocMDF              BOOLEAN,
  idPres                BOOLEAN,
  assocFragEnab         BOOLEAN,
  sDESAP                SAP-ID,
  algorithmChoices      AlgorithmChoices,
  validityPeriod        [0] ValidityPeriod OPTIONAL,
  callingMACAddress     [1] MACAddress OPTIONAL,
  calledMACAddress      [2] MACAddress OPTIONAL,
  callingMDF            [3] OCTET STRING OPTIONAL,
  calledMDF             [4] OCTET STRING OPTIONAL,
                        -- provided by the responder if assocMDF is TRUE
                        -- and the responder wished to use an MDF value
  securityLabel         [5] SecurityLabel OPTIONAL  }
```

The SAP-ID and MACAddress ASN.1 types are imported from IEEE 802.10f-1993. The others are defined below:

```
AlgorithmChoices  ::=  SEQUENCE OF AlgorithmPair

AlgorithmPair  ::=  SEQUENCE  {
padding                 BOOLEAN,
confidAlgorithm         [0] AlgorithmIdentifier OPTIONAL,
integAlgorithm          [1] AlgorithmIdentifier OPTIONAL  }
                        -- If the confidentiality Boolean is true,
                        -- then the confidAlgorithm must be present.
                        -- If the integrity Boolean is true,
                        -- then the integAlgorithm must be present.

ValidityPeriod  ::=  SEQUENCE  {
  notBefore             GeneralizedTime,
  notAfter              GeneralizedTime  }

SecurityLabel  ::=  CHOICE  {
  implicit              RegisteredSecurityLabel,
  explicit              RegisteredSecurityLabelSet  }

RegisteredSecurityLabel  ::=  SEQUENCE  {
  tagSetName            OBJECT IDENTIFIER,
  sdeSecurityLabel      AssociationLabelSet  }

RegisteredSecurityLabelSet  ::=  SEQUENCE  {
  tagSetName            OBJECT IDENTIFIER,
  sdeSecurityLabelSet  SET OF AssociationLabelSet  }

AssociationLabelSet  ::=  SEQUENCE OF SecurityTag

SecurityTag   ::=  CHOICE  {

    -- Type 1 - for restrictive security attributes
      restrictivebitMap   [1] IMPLICIT SEQUENCE  {
      securityLevel       SecurityAttribute,
      attributeFlags      BIT STRING  }

    -- Type 2 - security attributes by number
      enumeratedAttributes[2] IMPLICIT SEQUENCE  {
      securityLevel       SecurityAttribute,
      attributeList       SET OF SecurityAttribute  }

    -- Type 5 - all security attributes in the range(s)
      rangeSet            [5] IMPLICIT SEQUENCE  {
      securityLevel       SecurityAttribute,
      rangeList           SET OF SecurityAttributeRange  }

    -- Type 6 - for permissive security attributes
      permissivebitMap    [6] IMPLICIT SEQUENCE  {
      securityLevel       SecurityAttribute,
      attributeFlags      BIT STRING  }
    -- Type 7 - format specified via registration
      freeFormField     [7] IMPLICIT ANY DEFINED BY tagSetName  }

SecurityAttributeRange  ::=  SEQUENCE  {
      upperBound          SecurityAttribute,
      lowerBound          SecurityAttribute  }

SecurityAttribute  ::=  INTEGER (0..ub-SecurityAttribute)

ub-SecurityAttribute  ::=  16383
```

3E.2 Example SDE attribute negotiation

For purposes of example, assume that the initiator wants to use confidentiality and integrity, but the initiator is willing to settle for confidentiality only. For confidentiality, the initiator insists on DES-CBC. If confidentiality and integrity are both used, the initiator is willing to use DES-CBC with either SHA-1 or MD5.

The initiator would construct a SEQUENCE of one SDEAttrs that contains the following:

```
SDEAttrs ::=          SEQUENCE  {
  assocMDF              FALSE,
  idPres                FALSE,
  assocFragEnab         TRUE,
  sDESAP                "Default",
  algorithmChoices  { { TRUE, [0] { 1 3 14 3 2 7 }, [1] { 1 3 14 3 2 15 } }
                        -- DES-CBC with SHA-1
          { TRUE, [0] { 1 3 14 3 2 7 }, [1] { 2 2 840 113549 2 5 } }
                        -- DES-CBC with MD5
                        { TRUE, [0] { 1 3 14 3 2 7 } } }
                        -- DES-CBC without integrity
  securityLabel         -- absent   --
  validityPeriod        -- absent   --
  callingMACAddress     [1] 0x0080c880c435,
  calledMACAddress      [2] 0x0080c880abcd,
  callingMDF            -- absent   --
  calledMDF             -- absent   --
}
```

The SEQUENCE of one SDEAttrs is transferred to the responder as part of the offered-attrs security exchange item.

The responder selects the first alternative in the SEQUENCE that is acceptable. In this case there is only one to choose from. Then, the responder returns the selected SDEAttrs in the accepted-attrs security exchange. For purposes of example, assume that the responder desires both confidentiality and integrity. Also, assume that DES-CBC with SHA-1 is an acceptable combination. The following SDEAttrs is transferred to the initiator as part of the accepted-attrs security exchange item.

```
SDEAttrs ::=          SEQUENCE  {
  assocMDF              FALSE,
  idPres                FALSE,
  assocFragEnab         TRUE,
  sDESAP                "Default",
 algorithmChoices { { TRUE, [0] { 1 3 14 3 2 7 }, [1] { 1 3 14 3 2 15 } } }
                        -- DES-CBC with SHA-1
  securityLabel         -- absent   --
  validityPeriod        -- absent   --
  callingMACAddress     [1] 0x0080c880c435,
  calledMACAddress      [2] 0x0080c880abcd,
  callingMDF            -- absent   --
  calledMDF             -- absent   --
}
```

To order IEEE standards…

Call 1. 800. 678. IEEE (4333) in the US and Canada.

Outside of the US and Canada:
1. 732. 981. 0600

To order by fax:
1. 732. 981. 9667

IEEE business hours: 8 a.m.–4:30 p.m. (EST)

For on-line access to IEEE standards information…

Via the World Wide Web:
http://standards.ieee.org/

Via ftp:
stdsbbs.ieee.org

Via a modem:
1. 732. 981. 0035

ISBN 1-55937-955-3