NISTIR 4983

A Study of OSI Key Management

Roberto Zamparo

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Computer Systems Laboratory
Computer Security Division
Gaithersburg, MD 20899

NIST

NISTIR 4983

A Study of OSI Key Management

Roberto Zamparo

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Computer Systems Laboratory
Computer Security Division
Gaithersburg, MD 20899

November 1992

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

ABSTRACT

For communications between computer systems to be useful in many environments, the systems and their communications must be secure. One prerequisite to secure communications is the management of keying material needed by the underlying cryptographic mechanisms that provide security. This report addresses key management as it applies to communications protocols based on the Open Systems Interconnection (OSI) architecture. It contains a criteria and model of OSI key management that allows schemes based on both secret key and public key cryptography to be incorporated. The report reviews significant issues of OSI key management and presents a generic protocol that resolves a majority of them. The abstract syntax notation (ASN.1) is used to specify the protocol. An example of how registration of ASN.1 protocol modules can be used to support algorithm specific security objects is also given.

Keywords: OSI, Key Management, Public Key Cryptography, Secret Key Cryptography, Computer Network Security, ASN.1

PREFACE

This report is the result of my one year stay as a guest researcher at the National Institute of Standards and Technology, Computer Security Division, Gaithersburg, Maryland, U.S.A. My employer, Swedish Telecom, funded the one year stay. The Department for Research and Development of Swedish Telecom, where I was working in the Data Communications group, is now part of Telia Research. Telia Research is a research center owned by Swedish Telecom.

The first four months of my stay were devoted to the study of key management, the next four months to the making of the protocol definitions appearing in Annex B of this paper and during the last four to the editing of this report. Of course, there were many iterations made during the one year period. Chapter 11 entitled Areas Not Covered, lists what I felt should have been included in the report but were not. The areas listed in Chapter 11 could not fit into the one year program.

During my one-year period in the U.S.A. I participated at the IEEE 802.10 working group meetings. Some of the topics in this paper originate from those meetings.

In this paper I have used the Transport Layer Security Protocol (TLSP) in most of the examples illustrating problems related to security protocols and Diffie-Hellman for illustrating most of the problems related to key exchanges. It is not my intention to endorse either of them. I chose TLSP because I participated in a prototype implementation of TLSP in Sweden and therefore it became the security protocol with which I am most familiar. For the prototype implementation we wanted to use software developed within the Swedish Information Technology Program, IT4, which restricted us to implement either a layer three or layer four security protocol. At that point in time we regarded TLSP to be the most mature working draft, the Network Layer Security Protocol (NLSP) did not exist at the time. There were three proposals for a layer three security protocol: the English EESP, the American SP3 and the Canadian SPX. A layer two solution, as proposed by the IEEE 802.10 Security Data Exchange (SDE) security protocol, was never considered.

I realized that because of time limitations I should not be able to make protocol specifications including both an asymmetric and symmetric form of key management. I chose to concentrate on the asymmetric part. I then selected Diffie-Hellman as the algorithm to use for illustration purposes. I would emphasize that these decisions were made entirely by me and do not reflect any views that the NIST Computer Security Division may have on the issues: asymmetric versus symmetric form of key management and the choice of specific algorithms. As the reader will notice, one of the main concerns in this paper is how to achieve algorithm independence.

ACKNOWLEDGMENTS

The author wishes to express thanks to the following publishers and persons:

John Wiley & Sons and Prentice-Hall which gave me the permission to reprint material from the books Security For Computer Networks and The Open Book respectively.

George Endersz, manager of R&D Information Security earlier at Swedish Telecom and presently at Telia Research, who initiated from Swedish Telecom´s part the cooperation with NIST on key management in open systems.

Ragnhild Nylund, Telia Research, who has edited this report.

Robert Rosenthal, who is manager of the protocol security group at NIST, made it possible for me to join as a guest scientist the group.

Russell Housley, Xerox Special Information Systems, who reviewed this report and provided me with a lot of useful and valuable comments on technical aspects, as well as suggestions on the outline of this paper. It was Russell Housley who gave me the initial information about the IEEE 802.10 key management activity.

Finally, thanks are above all due to Wayne Jansen who is working with the NIST protocol security group. He has continuously, over a long period of time, read and commented on this paper providing many useful and valuable comments. He has very patiently corrected and improved my English.

1 INTRODUCTION

1.1 BACKGROUND

This paper deals with key management in the Open Systems Interconnection (OSI) architecture. The well known OSI seven layer model is formally defined in [ISO 7498]. The aim of OSI is to allow, with a minimum of technical agreements outside the interconnection standard, the interconnection of computer systems:

- from different manufactures;
- under different managements;
- of different levels of complexity;
- of different ages.

The Security Architecture document [ISO 7498-2] defines the general security related architectural elements required for the protection of communications. [ISO 7498-2] describes in which layer a particular security service can be provided and it also describes the security mechanisms employed to implement the various security services. The Guide To Open System Security working draft document [ISO/SC21 6167] describes in detail how OSI security standards are organized. At the moment key management standardization within OSI lags behind the standardization of security protocols. Therefore, when the standardization of security protocols reach a mature status, key management unavailability may present a bottleneck.

[ISO 7498-2] describes key management as: "The generation, storage, distribution, deletion, archiving and application of keys in accordance with a security policy." Thus, there are many aspects of key management that are outside the scope of OSI. The following description gives an informal explanation of the need for key management in OSI. The OSI aspects of key management are mainly concerned with the establishment of encryption keys and attributes for cryptographic mechanisms within peer communicating security protocol entities in order to communicate securely. An example of secure communications is to employ a confidentiality and integrity service provided by a lower layer security protocol. Two such OSI security protocols are the Transport Layer Security Protocol (TLSP) [ISO 10736] and the Network Layer Security Protocol (NLSP) [ISO 11577]. Communicating parties need to establish the keys and attributes automatically (that is, electronically), in order to avoid the expense associated with manual distribution methods.

The kinds of keys and associated attributes include:

- The *encryption algorithm* to implement the confidentiality service provided by the security protocol entity.
- The *encryption key* used by the encryption algorithm mentioned above.
- The *initialization vector* used to initialize values needed by the cryptographic algorithm.
- The *integrity algorithm* to implement the integrity service provided by the security protocol entity.
- The *encryption key* used by the integrity algorithm mentioned above.
- The *security label*[1] syntax that is going to be used by the security protocol entity.

[1] The reader is referred to [SEC LABEL90] and [SEC LABEL91].

- The allowed *security level range* of the data to be protected by the security protocol entity. When two security protocol entities communicate the security level of the protected data is indicated by a security label.

Note, in security standards the above mentioned encryption keys are often called *session keys*, *data keys*, or *traffic encryption keys*. In this paper the term session keys is not used because a reader familiar with the OSI model but not with OSI security may relate session keys to the session layer.

Figure 1.1:1 illustrates the following key exchange scenario:

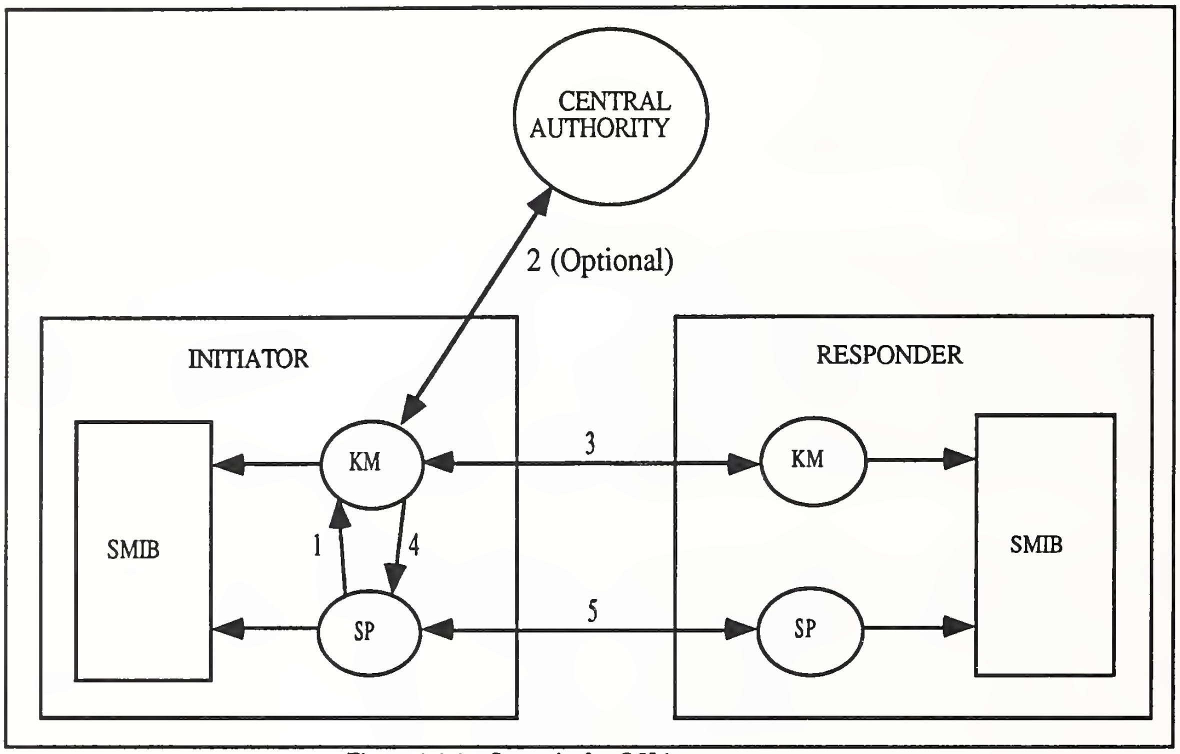


Figure 1.1:1 Scenario for OSI key management

1. The initiating security protocol (SP) entity informs its key manager that it wants to establish keys and attributes with the identified peer entity.
2. This step may or may not be performed. Whether it is performed depends on the key management scheme employed, this step is often associated with a key management scheme based on a central authority (third party); whereby the central authority may generate the keys for the key manager, or alternatively the keys may instead be generated by the key manager but need to be passed to the central authority for notarization[2].
3. If the central authority has not been involved the key managers typically establish keys and attributes directly with one another, and store the result in the Security Management Information Base (SMIB). A description of SMIB is given in the Security Architecture document [ISO 7498-2]. When the central authority has been involved the initiating key manager distributes the keys to the responding key manager. As for the other case, the attributes established together with the keys are stored in the SMIB.

[2] Annex A provides more details about notarization.

4. The key manager at the initiating side informs the SP entity that the keys and attributes have been established.
5. The two security protocols can communicate with each other, the keys and attributes are retrieved from the SMIB.

The key manager has to provide the security protocol with key materials, whose origin, integrity and in the case of secret keys confidentiality, are guaranteed. The challenge is to establish the traffic encryption keys (TEKs) over an insecure network.

1.2 PLACE OF KEY MANAGEMENT IN THE OSI ARCHITECTURE

The OSI Security Architecture document [ISO 7498-2] does not mandate the key management functionality to reside in a particular layer. [ISO 7498-2] regards key management to be part of security management. If the key management functionality is placed in the application layer, the following alternatives are possible:

- To have key management as a separate Application Service Element (ASE).
- To have key management based on CMIS/CMIP [ISO 9595] [ISO 9596].

At a first glance alternative number two seems to be the preferred one, because CMIS/CMIP is used to transfer management information. In CMIP there is an asymmetric relationship between the manager and the agent. It is probably more natural to view a key exchange to occur between two equal parties. However, other application layer protocols may be used for special management purposes. One possibility is therefore to specify a layer seven key management protocol. This key management protocol can be an ASE built upon the ROSE ASE (as, for example, CMIP is based on ROSE) or be an ASE independent of ROSE (as, for example, FTAM).

[ISO 7498-2] also states that exchange of working keys for use during an association is a normal layer protocol function. This approach is used by NLSP [ISO 11577]. Cryptographic keys to be used in an NLSP security association can be established when the security association is created on-line. Security associations are explained in section 5.18.

1.3 KEY MANAGEMENT ACTIVITIES

Existing key management standards and standard activities include:

- Secure Data Network System (SDNS);
- ANSI X9.17;
- ANSI X9.28;
- ISO/CD 11166;
- Standards for Interoperable LAN Security (SILS);
- ISO SC 27;
- Network Layer Security Protocol (NLSP).

1.3.1 Secure Data Network System (SDNS)

The SDNS project produced a family of standards intended to facilitate secure communication between open systems. The SDNS architecture [TATE87] mandates a layer seven OSI key management protocol. The SDNS is designed for protection of both classified and unclassified information. The cryptographic algorithms as well as associated details of required protocol control information for classified key management have not been made publicly available.

Lower layer security protocols (SP) for the Network (SP3) and Transport (SP4) layers also originate from the SDNS project. They form the basis for the draft OSI security proposals known respectively as NLSP-CL and TLSP.

The SDNS key management protocol [SDNS 90-4262] is of special interest because it is the first available layer seven key management protocol definition.

1.3.2 ANSI X9.17

X9.17 [ANSI X9.17] is a key management standard used by financial institutions. [ANSI X9.17] specifies the Data Encryption Standard (DES) [FIPS 46-1] as the algorithm to protect the data keys and is, thus, based on symmetric techniques. It cannot directly be used as an OSI key management protocol due to a number of incongruities it has with the OSI architecture. For example, an OSI application protocol must be specified by using the ASN.1 notation, it must make use of the Application Control Service Element (ACSE) [ISO 8649] [ISO 8650] to establish an application association and has to rely on services provided by the presentation layer. [ANSI X9.17] does not do any of this. Furthermore, [ANSI X9.17] is not aware of what an OSI security protocol is, so it cannot establish the special attributes needed by the security protocol. [ISO 8732] is similar to [ANSI X9.17] but does not presume the use of a specific symmetric cryptographic algorithm.

1.3.3 ANSI X9.28

Two subscribers, A and B, are, in [ANSI X9.17], served by the same Key Distribution or Key Translation Center. In the X9.28 standard [ANSI X9.28] two subscribers share a Multiple Center Group (group) consisting of two or more centers. The group offers its subscribers a service equivalent to the one provided by the [ANSI X9.17] Key Distribution or Key Translation Center.

In contrast to [ANSI X9.17] this standard proposal is not based on a specific algorithm, although it is based on symmetric cryptographic techniques. Due to the same reasons as for [ANSI X9.17], this standard cannot be used for an OSI environment.

1.3.4 ISO/CD 11166

This key management standard proposal is an asymmetric version of ISO 8732. Instead of using symmetric key cryptography to encipher the key material, encipherment is accomplished by asymmetric key cryptography. The standard is not directly applicable as a future OSI key management protocol for the same reasons as [ANSI X9.17]. [ISO 11166] addresses only the case where there is one Key Certification Center (CKC), an annex including Multiple Certification Centers will be added at a later stage.

1.3.5 Standards for Interoperable LAN Security (SILS)

SILS key management is an ongoing standardization activity within IEEE 802.10. The goal is to standardize a general purpose key management protocol which supports the IEEE 802.10 Secure Data Exchange (SDE) security protocol [IEEE P802.10B] as well as other security protocols. The approach taken is to have the key management protocol as an OSI layer seven application protocol. Three types of key management schemes are supported:

- Key management based on asymmetric cryptography.
- Key management based on symmetric cryptography.
- Key management based on manual methods.

So far the work has been done mainly on the asymmetric part of the protocol.

1.3.6 ISO SC 27

ISO JTC 1 SC 21 has assigned the responsibility for the development of the framework for key management to SC 27. Work began in April 1991 and it had not produced any mature working documents when this report was written. SC 27 WG 1 is responsible for the framework document and WG 2 for key management techniques. Note that the framework just gives guidelines to protocol designers. After the framework is available, the work to specify the protocol still remains.

1.3.7 Network Layer Security Protocol (NLSP)

The Network Layer Security Protocol (NLSP) [ISO 11577] includes both a connection oriented part (NLSP-CO) and a connectionless part (NLSP-CL). NLSP provides the option of performing a key exchange. It is not clear why the approach to have key management as part of the security protocol was chosen. One motive could be that at the moment there is no layer seven key management protocol and some alternative is needed to provide the security protocol with keys and attributes. Another motive could be that the designers of the protocol prefer to have key management outside the application layer in a less general protocol, tailored to the layer. However, NLSP does not preclude having key management at a layer other than that of the security protocol. It is possible to establish key and attributes through a layer seven key manager.

2 CRITERIA FOR OSI KEY MANAGEMENT

2.1 GENERAL CRITERIA FOR OSI KEY MANAGEMENT

In OSI Key Management the following requirements should be satisfied:

- The key management protocol has to be algorithm independent and support both public and secret cryptographic systems.
- It must be possible to perform key management across security domain boundaries.
- The key management protocol shall support different types of security protocols with keys and security association attributes.
- The same form of key management should be applicable across a wide range of networking environments (e.g., to both LAN and WAN environments).

2.1.1 Algorithm independence

A security domain must have the ability to choose the algorithms consistent with its underlying security policy. In order to achieve this, there must be no assumptions in the key management protocol that specific cryptographic algorithms have to be used. Algorithms include those required for confidentiality, integrity and key exchange. It is assumed that authentication is included in the key exchange algorithm.

No OSI key management standard can be based on specific algorithms, since it is a practice in international security standards not to specify a specific algorithm. Some organizations may have private algorithms that they want to keep secret. Another motive for keeping the protocol independent of algorithms is that if a particular algorithm is broken, the protocol can be adapted to another algorithm. The ideal solution is a general purpose key management protocol, which can be configured with different types of cryptographic algorithms and support the full range of key management schemes.

2.1.2 Operations across security domains

The Key Management Protocol must allow two key managers belonging to different security domains to perform a key exchange. This capability may be realized in a variety of ways ranging from direct exchange between two parties to more indirect means involving multiple parties.

2.1.3 Support of a variety of security protocols

The Key Management Protocol must support security protocols that operate at different layers of the OSI reference model. The key management functionality therefore needs to be located at only one layer in the reference model offering a common service. TLSP [ISO 10736], NLSP [ISO 11577] and SDE [IEEE P802.10B] are examples of different kinds of security protocols that can be supported by a common key management protocol.

2.1.4 Support across a range of networking environments

The key management protocol must be usable across a wide range of networking environments in particular Local Area Network (LAN) and Wide Area Network (WAN) environments. Solutions only applying to one environment but not to to the other must be avoided. One such problematic solution is where there are assumptions about the synchronization of clocks. In some key management schemes this may be a prerequisite, for example, [DENN81] describes a key exchange where the synchronization of clocks is a necessity. Such a scheme is probably easier to implement in a LAN environment.

2.2 GENERAL REQUIREMENTS

The requirements listed in the previous paragraphs are not general requirements for key management. The requirements listed are specific for an OSI environment. Requirements of a more general nature are listed in chapter 4 of [ANSI X9.17] and in the article Principles Of Key Management [FUMY90]. These requirements stipulate that:

- a data network may be expanded, therefore, the key management system must be scalable ranging from a small to a large number of systems;
- the key management architecture shall support any party initiating a communication with any other party;
- key security and integrity shall be ensured;
- the key material should have a limited lifetime based on time, use, or other criteria;
- the use of couriers should be kept at a minimum;
- the need of secure channels should be minimized;
- there shall be division of responsibilities so that no person has complete information about important material (e.g., complete copies of a key encrypting key[1]);
- a traffic encryption key shared between a communicating pair shall be secured from third party usage (except for a key center[2]);
- a traffic encryption key can be used for either confidentiality or integrity but not both. Exception from this rule is that a traffic encryption key for confidentiality may be used to guarantee integrity of Cryptographic Service Messages (CSMs) during the distribution[3] phase;
- the compromise of any key shared between two communicating pairs shall not compromise any third party;
- a key used between any communicating pair shall not intentionally be used between any other communicating pair.

[1] Annex A explains what a key encrypting key is.

[2] Synonymous to the Central Authority described in section 1.1.

[3] CSMs can be viewed to correspond to the PDUs of an OSI environment.

3 KEY MANAGEMENT MODEL

An object oriented model is used to describe OSI key management. The model is to the largest extent possible kept independent of a specific key management scheme. The model also has to be applicable to both asymmetric and symmetric forms of key management. In later chapters the model is used as a meta-model, i.e., the model we use when we are talking about key management.

3.1 TOP LEVEL VIEW

The following figure shows the top level view of the model:

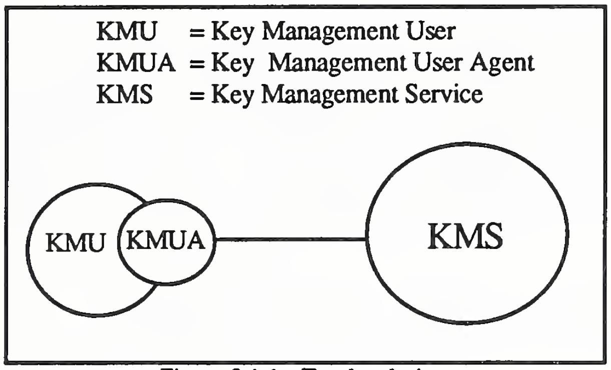

KMU = Key Management User
KMUA = Key Management User Agent
KMS = Key Management Service
KMU
KMUA
KMS

Figure 3.1:1 Top level view

The model consists of three object types which represent communication entities, the Key Management User object (KMU), Key Management User Agent object (KMUA) and Key Management Service object (KMS). The line between the objects represents an OSI instance of communication. A Key Management User employs the Key Management Service through its Key Management User Agent. The KMU may represent a security protocol, such as: TLSP [ISO 10736], NLSP [ISO 11577] or SDE [IEEE P802.10B].

3.2 DECOMPOSITION OF THE KEY MANAGEMENT SERVICE

The following figure shows the decomposition of the Key Management Service object:

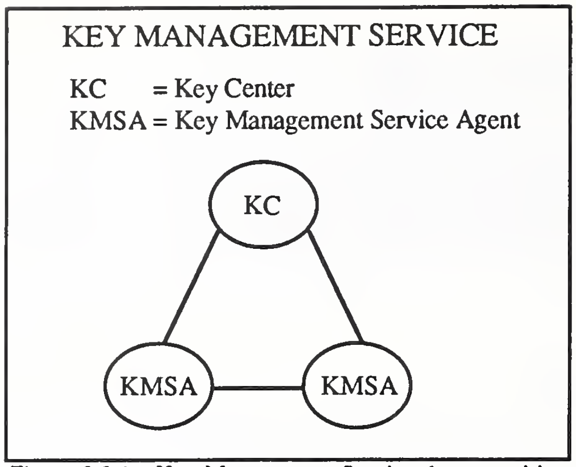


Figure 3.2:1 Key Management Service decomposition

The Key Management Service consists of two Key Management Service Agent objects and a Key Center object. The mission of the KMSAs is to establish the keys and attributes to be used by the two communicating security protocol entities. Objects may be collocated within an end-system thereby avoiding OSI communication through intra-system communication capabilities. It is not clear whether in practice the initiating system always has the KMUA and KMSA collocated. In the model it is possible to keep the KMSA and KMUA apart. If they are kept apart, the communication path between the KMUA and KMSA must be trusted or protected.

3.3 DECOMPOSITION OF THE KEY CENTER

The Key Center consists of a number of Key Center Agents (KCA) related to each other hierarchically:

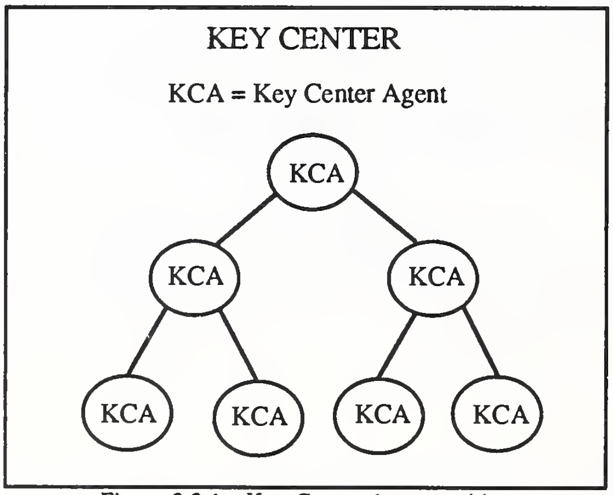


Figure 3.3:1 Key Center decomposition

Each key center agent can be viewed as residing within the juristdiction of the security domain of its immediate predecessor in the KCA hierarchy and deriving its authority from it. This is illustrated in the figure below:

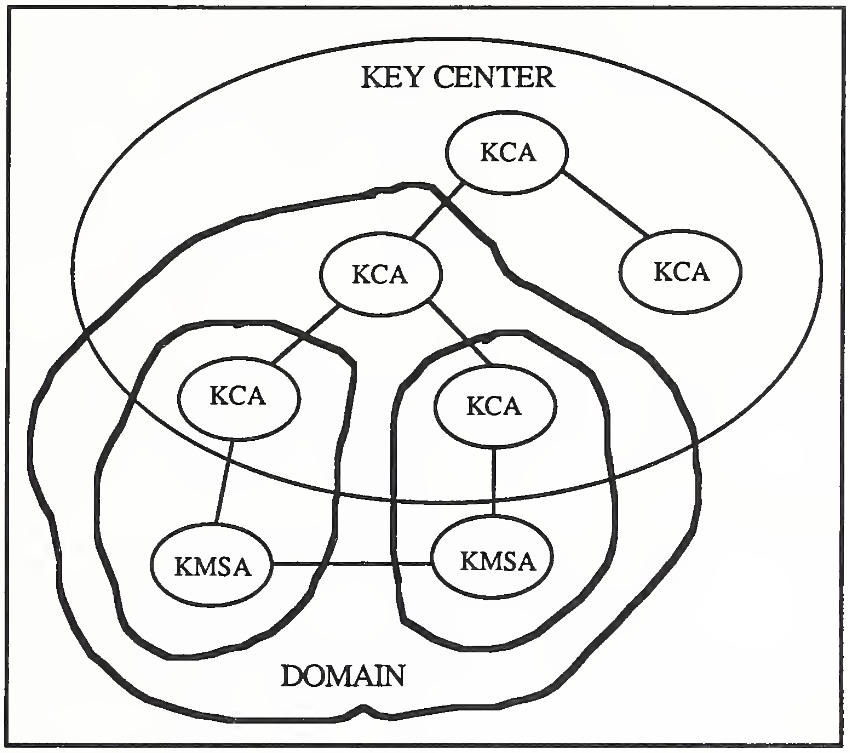


Figure 3.3:2 Relationship between Security Domain and KCA

4 PHASES IN TRAFFIC ENCRYPTION KEY MANAGEMENT

The phases that cooperating key management server agent entities go through during the lifetime of traffic encryption key material is exemplified by the approach taken in SDNS. SDNS key management can be divided into the following phases:

- Key Establishment.
- Service Negotiation/Cryptographic compatibility.
- Key Usage.
- Key Destruction.

During the Key Establishment phase, traffic encryption keys are created. As mentioned earlier the algorithm used in SDNS is classified. An example of a publicly available algorithm which may be used is Diffie-Hellman (see Annex A for further information). In SDNS for the key establishment phase the New Key Request (Nkrq) and New Key Response (Nkrs) PDUs are used.

During the Service Negotiation and Cryptographic Compatibility phase the two communication parties demonstrate to each other that they are in possession of the right traffic encryption keys. In SDNS this is accomplished by encrypting the Security Service Request (Ssrq) and Security Service Response (Ssrs) PDUs using the newly established key. Anyone attempting to replay a previous executed key exchange would fail during this phase. The Ssrq and Ssrs PDUs are also used to negotiate the attributes for the security services which are needed for a specific security protocol.

The traffic encryption keys are used to protect the data transferred between the security protocol entities. The key management protocol entity is not involved during this phase, unless the cryptoperiod of the traffic encryption key expires and the traffic encryption key must be replaced or updated.

The Key Destruction phase occurs when the cryptoperiod of the key expires, or in the case of TLSP (SP4C) used on a per (transport) connection basis when the transport connection is closed.

Since the algorithm used to do the SDNS key exchange is classified no details are given about how the authentication is done. Annex A gives an example of how to perform the authentication when Diffie-Hellman is used.

5 ISSUES IN KEY MANAGEMENT

In this chapter two forms of key management are considered:

- A symmetric form of key management, where the parties share a Key Encryption Key (KEK) with each other or a Key Distribution Center, alternatively a Key Translation Center.
- An asymmetric form of key management which is certificate-based. In this form of key management the parties are given certificates from a trusted authority, the Certification Authority (CA).

These two forms of key management can be considered the state-of-the-art.

Key management standards outside of OSI are considered in this chapter, since an OSI key management protocol could be based on principles defined in these standards.

In the following sections the term CA is sometimes used and sometimes the term KCA, the intention is to use the term CA in general discussions while the term KCA is used when referring to the key management model defined earlier.

This paper tries to limit the use of the term credentials because in some security documents its use is in conflict with the definition of credentials as defined in the Security Architecture [ISO 7498-2]. The definition of [ISO 7498-2] is: "Data that is transferred to establish the claimed identity of an entity." To avoid mentioning the term credentials the phrase "private key and certificate, alternatively certificate" is used in this paper. For example, the statement that the KCA delivers the credentials to a KMSA would be in conflict with [ISO 7498-2], because the delivery could include the private key of the KCA. According to the definition in [ISO 7498-2] the private key does not need to be part of the credentials. When this paper uses the term credentials, the meaning of credentials depends on the context.

5.1 HOLDER OF THE CERTIFICATE

An important issue in an asymmetric form of key management is who should be the holder of the certificate. The Authentication Framework [ISO 10181-2] gives as an example the following entities to authenticate:

- a physical entity;
- a human user;
- an application entity.

In reality, therefore the following entities could be the holder of the certificate:

- an end system (physical entity);
- a security device within the end system (physical entity);
- a human user;
- the key management application entity (application entity).

In Standard for Interoperable LAN Security (SILS) key management the holder of the certificate is the Key Management Application Process. When authentication occurs, it is the key management application processes that authenticate each other. Thus, a key management application process residing in system A can be convinced of communicating with an key management application process residing in system B. And not with another key management application process claiming to reside at system B.

[ISO 10181-2] defines an authentication certificate as: "A particular type of certificate, certified by a trusted authority, which may be used for authentication." However, certain key management schemes may use the public key included in the certificate to encrypt key material. [ISO 10181-2] mentions two types of authentication certificates:

- on-line authentication certificates;
- off-line authentication certificates.

The definition of an on-line certificate is: "An on-line authentication certificate is created by a trusted third party by direct request of a claimant." One example of this type of certificate is the Privilege Attribute Certificate (PAC) described in [ECMA-138]. The definition of an off-line certificate is: "An off-line authentication certificate binds an identity to a cryptographic key. It is created by an authority, without either the claimant or verifier needing to interact with the authority." The certificates defined in the Directory Authentication Framework [ISO 9594-8] are off-line certificates. The certificates mentioned in this paper are also off-line certificates.

5.2 ORGANIZATION OF THE CERTIFICATION AUTHORITIES

The organization of certification authorities (CAs) is an issue for the asymmetric form of key management. The first design issue is whether just one or several certification authorities should be supported. An OSI environment consists of users affiliated to different organizations and residing across national boundaries; to have only one certification authority in such an environment seems therefore unrealistic. This leads to the second design issue: how should the certification authorities be related to each other? Different models of how to organize the certification authorities are reviewed later in this paper. In our model of key management a KCA possesses the functionality provided by a certification authority. Annex A explains in further detail the nature of a certification authority (CA) and certificates.

This section makes an attempt to use the existing infrastructure provided by OSI, i.e., the Directory Authentication Framework [ISO 9594-8]. The other alternative which is not covered in depth, is that a security domain creates its own infrastructure. For example, the Key Management Center (KMC) in SDNS can be regarded as part of the infrastructure designated for a particular security domain.

5.2.1 Organization of certification authorities in SDNS

According to Architectural Model Of The SDNS Key Management Protocol [LAMB88] the single certification authority in the SDNS world is the KMC. If we try to interpret SDNS in our model for key management, SDNS can be viewed as a single level hierarchy.

5.2.2 Organization of certification authorities in the Directory standards

The Directory Authentication Framework [ISO 9594-8], which is one of the parts constituting the OSI Directory standards, indicates that the CAs could be arranged in a hierarchy. There can be two different kinds of certificates, CA certificates and user certificates. A CA certificate is a certificate where a CA certifies another CA. CAs must certify each other in order to make it possible for users whose public keys are certified by different CAs to authenticate each other. When the CAs are arranged in a hierarchy, there can be both forward certificates and reverse certificates. A forward certificate is a certificate where a CA certifies the public key of another CA or a user. The certified CA or user is located in child position to the certifying CA. A reverse certificate is a certificate where a child (subordinate) CA certifies its parent (immediate superior) CA.

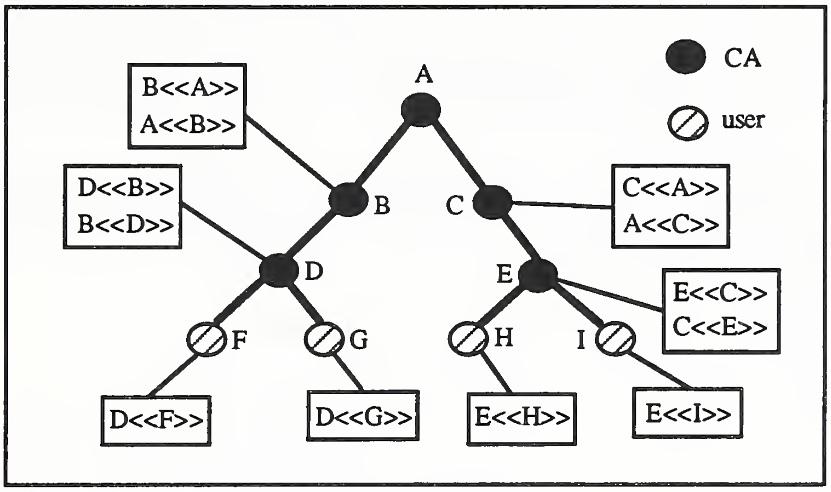


Figure 5.2.2:1 Hypothetical hierarchy

In the figure above, D<<B>>, indicates the reverse certificate for CA D and B<<D>> the forward certificate. The notation D<<B>> means a certificate issued by D for B. If, for example, user F has to authenticate user H, F needs to obtain the following certificates in addition to knowing the public key of D, D_p:

D<<B>>, B<<A>>, A<<C>>, C<<E>>, E<<H>>

This chain of certificates is called *certification path.* [ISO 9594-8] describes a certification path to be: "a list of certificates needed to allow a particular user to obtain the public key of another." User F, in possession of the public key of user H, can easily verify a digital signature created by H. Thus, the identity of the user can be authenticated.

[ISO 9594-8] describes the unwrapping of a certificate or certificate chain as an infix operation. The operator, •, has two operands: a public key of a CA and a certificate. For example, the operation performed by F to obtain the public key of H is:

Hp = Dp • D<<B>>, B<<A>>, A<<C>>, C<<E>>, E<<H>>

The operation consists of several steps, the first step is to obtain the public key of CA B. It is assumed that user F is in possession of the public key of CA D, Dp. The operation to obtain the public key of CA B is:

Bp = Dp • D<<B>>

When the next step occurs, F is in possession of the public key of CA B and has to obtain the public key of CA A:

Ap = Bp • B<<A>>

This is repeated until the public key of user H is obtained.

An important concept given in [ISO 9594-8] is: "...there may be no relationship between the Directory Information Tree (DIT) and the organization of the certification authorities." Thus, there is no prerequisite that the CAs have to be arranged in a hierarchy, [ISO 9594-8] permits also non hierarchical relationships. Another important concept given in [ISO 9594-8] is: "The precise method employed by users A and B to obtain certification paths A $\rightarrow$ B and B $\rightarrow$ A may vary. One way to facilitate this is to arrange a hierarchy of CAs, which may or may not coincide with all parts of the DIT[3] hierarchy." Thus, no assumptions are made about having the certification hierarchy to coincide with the naming hierarchy Therefore, even if a hierarchical relationship is assumed, there may be CAs included in the certification path whose distinguished names do not coincide with the distinguished name of the users performing an authentication. [RFC 1114] describes the problem in the following way:

> Unforgeable certificates are generated by certification authorities; these authorities may be organized hierarchically, though such organization is not required by X.509. There is no implied mapping between a certification hierarchy and the naming hierarchy imposed by the directory system naming attributes.

A non-hierarchical relationship is illustrated in the figure below:

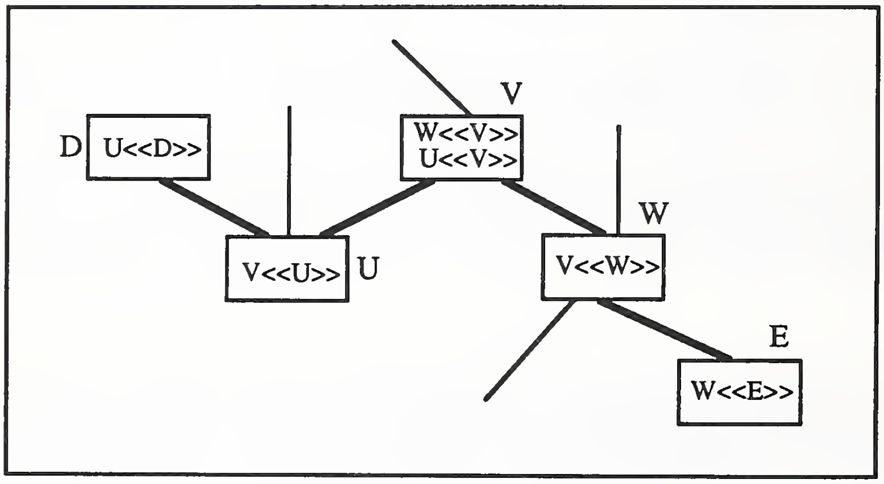


Figure 5.2.2:2 Non-hierarchical relationship

In order to establish a certification path between D and E in the figure above, a directed graph must be created. A criticism is that it might be difficult to determine the directed graph, and if it is possible to determine the directed graph the procedure employed may be too time-consuming. What is desirable is to be able to create a certification path by just knowing the other entities distinguished name. If the CAs are hierarchically related and there are CAs included in the certification path whose distinguished names do not coincide with the distinguished name of the users, this becomes harder to achieve. The following example attempts to illustrate the problem. Let us assume that Dennis has to verify a signed message sent by Russ and the distinguished names of Dennis and Russ are:

{c = US, o = Government, ou = Department of Commerce, ou = NIST, ou = CSL, cn = Dennis Branstad}

and

[3] Directory Information Tree

{c = US, o = Xerox, l = McLean, cn = Russell Housley}

Where:

c	=	Country Name
o	=	Organization Name
ou	=	Organizational Unit Name
l	=	Locality Name
cn	=	Common Name

The naming hierarchy and the CA hierarchy is shown in the figure below:

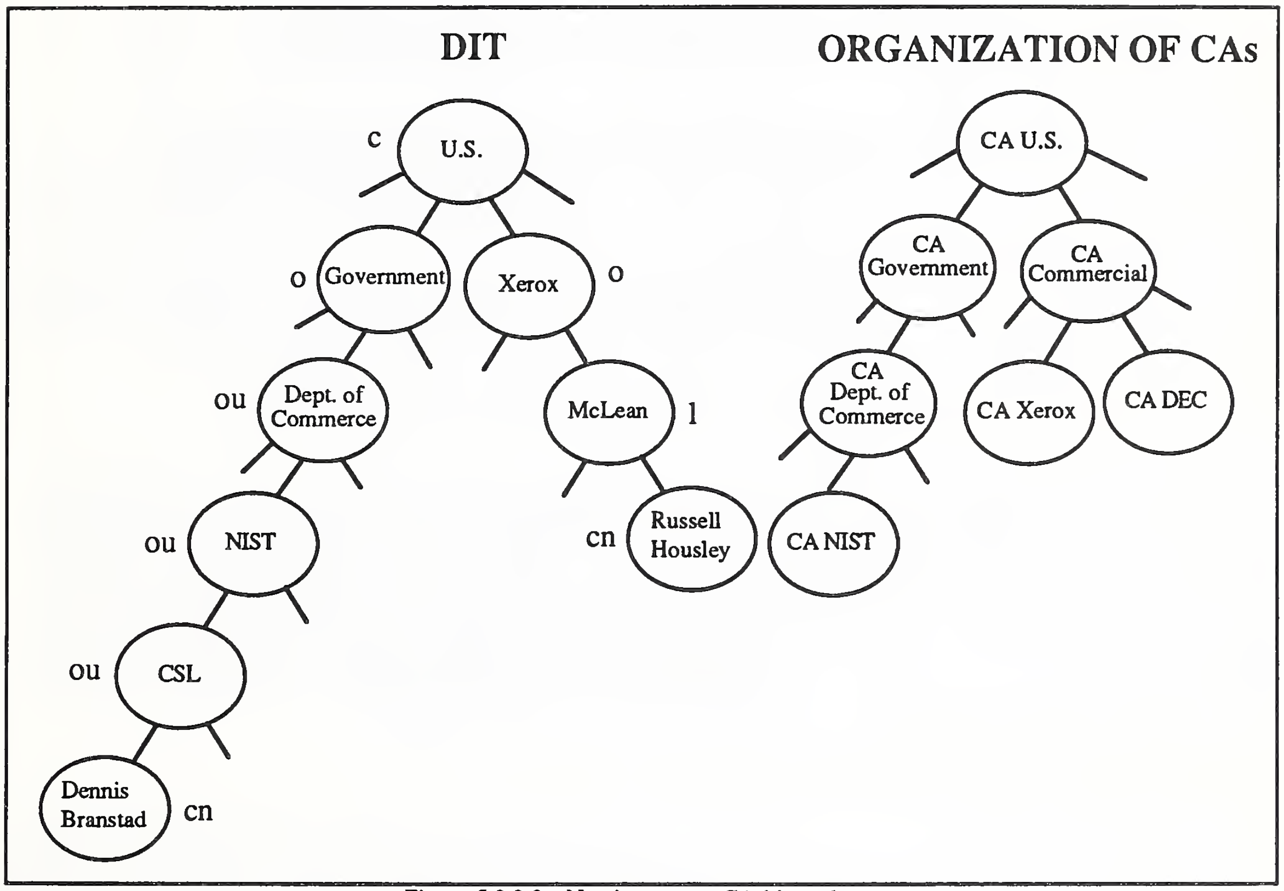


Figure 5.2.2:3 Naming versus CA hierarchy

In this example the CA whose distinguished name does not coincide with the distinguished name of the users is CA Commercial. The certification path needed for Dennis to verify a message signed by Russ is:

CA NIST <<CA Department of Commerce>>,
CA Department of Commerce<<CA Government>>, CA Government <<CA U.S.>>,
CA U.S.<<CA Commercial>>, CA Commercial<<CA Xerox>>,
CA Xerox<<Russell Housley>>.

Since the CA hierarchy cannot be derived from the distinguished name of the users, the problem is how to determine the common point of trust. The common point of trust is the lowest CA in hierarchy that is in common for both users. One possibility to create the certification path may be to follow the reverse certificates, starting with the reverse certificate issued by Dennis´ CA, CA NIST. From the different Directory entries are the reverse certificates retrieved and included in the certification path. This continues until the uppermost CA, CA U.S., is reached.

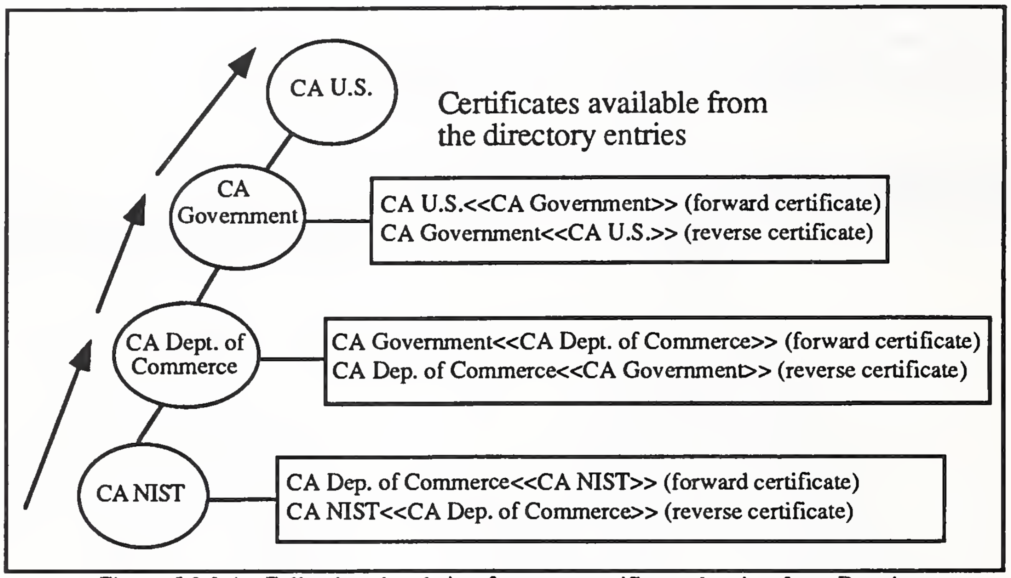

CA U.S.
Certificates available from the directory entries
CA Government
CA U.S.<<CA Government>> (forward certificate)
CA Government<<CA U.S.>> (reverse certificate)
CA Dept. of Commerce
CA Government<<CA Dept. of Commerce>> (forward certificate)
CA Dep. of Commerce<<CA Government>> (reverse certificate)
CA NIST
CA Dep. of Commerce<<CA NIST>> (forward certificate)
CA NIST<<CA Dep. of Commerce>> (reverse certificate)

Figure 5.2.2:4 Following the chain of reverse certificates leaving from Dennis

When CA U.S. is reached, it would be natural to transit from CA U.S. to CA Commercial. However, since Commercial is not visible from the users distinguished names; it is at this point unknown that CA Commercial is the second level CA included in the certification path originating with CA U.S.<<CA Commercial>> and ending with CA Xerox<<Russell Housley>>. Therefore, instead the reverse certificates originating with the reverse certificate of the CA which issued Russ´ certificate, CA Xerox, are followed. From the different Directory entries the reverse and forward certificates are retrieved. The forward certificates are included in the certification path. This continues until the common point of trust, CA U.S., is discovered.

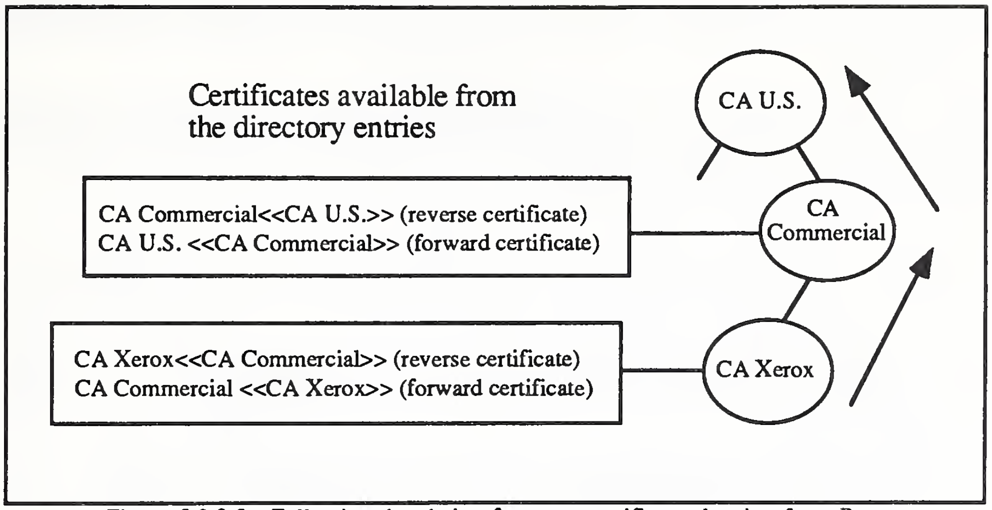

Certificates available from the directory entries
CA Commercial<<CA U.S.>> (reverse certificate)
CA U.S. <<CA Commercial>> (forward certificate)
CA Xerox<<CA Commercial>> (reverse certificate)
CA Commercial <<CA Xerox>> (forward certificate)
CA U.S.
CA Commercial
CA Xerox

Figure 5.2.2:5 Following the chain of reverse certificates leaving from Russ

As we will see in the next section, by letting the CA hierarchy coincide with the naming hierarchy, it becomes easier to find the certification path.

To make it possible to shorten the certification path, [ISO 9594-8] mentions that two certification authorities can cross-certify each other. The figure below gives an example of cross-certification (between E and F).

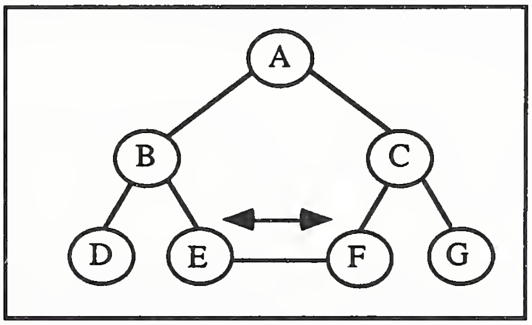


Figure 5.2.2:6 Example of cross-certificates

It is outside the scope of [ISO 9594-8] to specify how the public key of the certification authority a user trusts is delivered to the user. According to Figure 5.2.2:1 for user F it would be the public key of certification authority D. User F must be guaranteed the authenticity and integrity of this public key.

5.2.3 Organization of certification authorities in RFC 1114

This section tries to capture the ideas behind [RFC 1114]. [RFC 1114] specifies a subset of the Directory Authentication Framework [ISO 9594-8]. The subset chosen makes it easier to implement the functionality by letting the CA hierarchy coincide with the naming hierarchy. In [RFC 1114] this is described as a relationship between the naming hierarchy and the organization of the CAs. Primarily the intention is to have organizations act as certification authorities. Although [RFC 1114] is designed to support privacy enhanced mail (PEM), the standard is written so that it also can be used to support X.400 mail security facilities and X.500 Directory authentication facilities.

As mentioned previously, [RFC 1114] forces the CA hierarchy to coincide with the naming hierarchy. The following example tries to illustrate the CA hierarchy/DIT relationship, the figure below presents a portion of a hypothetical DIT:

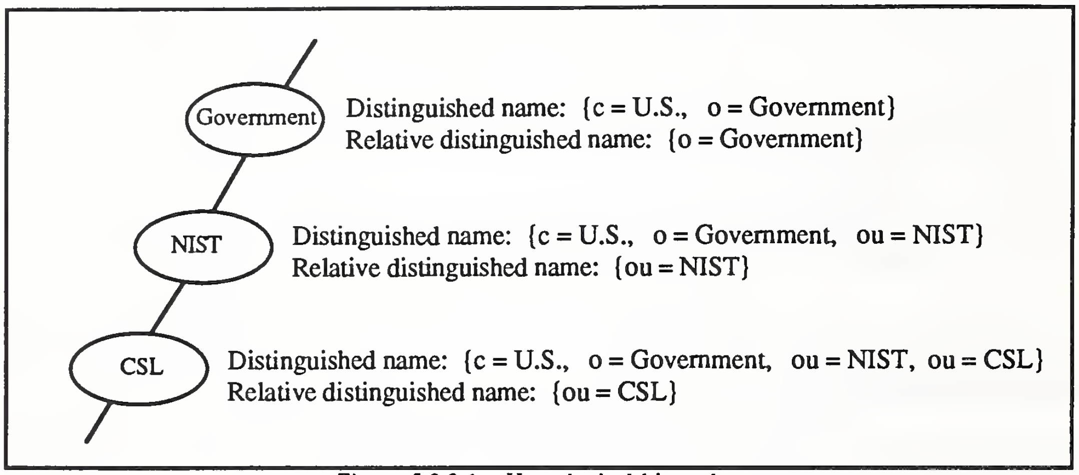

Government
Distinguished name: {c = U.S., o = Government}
Relative distinguished name: {o = Government}
NIST
Distinguished name: {c = U.S., o = Government, ou = NIST}
Relative distinguished name: {ou = NIST}
CSL
Distinguished name: {c = U.S., o = Government, ou = NIST, ou = CSL}
Relative distinguished name: {ou = CSL}

Figure 5.2.3:1 Hypothetical hierarchy

We assume that NIST and CSL both act as certification authorities. NIST is an immediate superior to CSL in the DIT. Therefore CSL is in possession of a forward certificate where the distinguished name of the issuer field is:

{c = U.S., o = Government, ou = NIST}

The subject field of the certificate contains the following distinguished name:

{c = U.S., o = Government, ou = NIST, ou = CSL}

The principle used in [RFC 1114] is that a certification authority should serve a large population of users, which is not a necessity in [ISO 9594-8]. Therefore [RFC 1114] precludes a user from acting as a CA. Figure 5.2.3:2 illustrates which of the entities represented in the DIT that are allowed by [RFC 1114] to act as certification authorities.

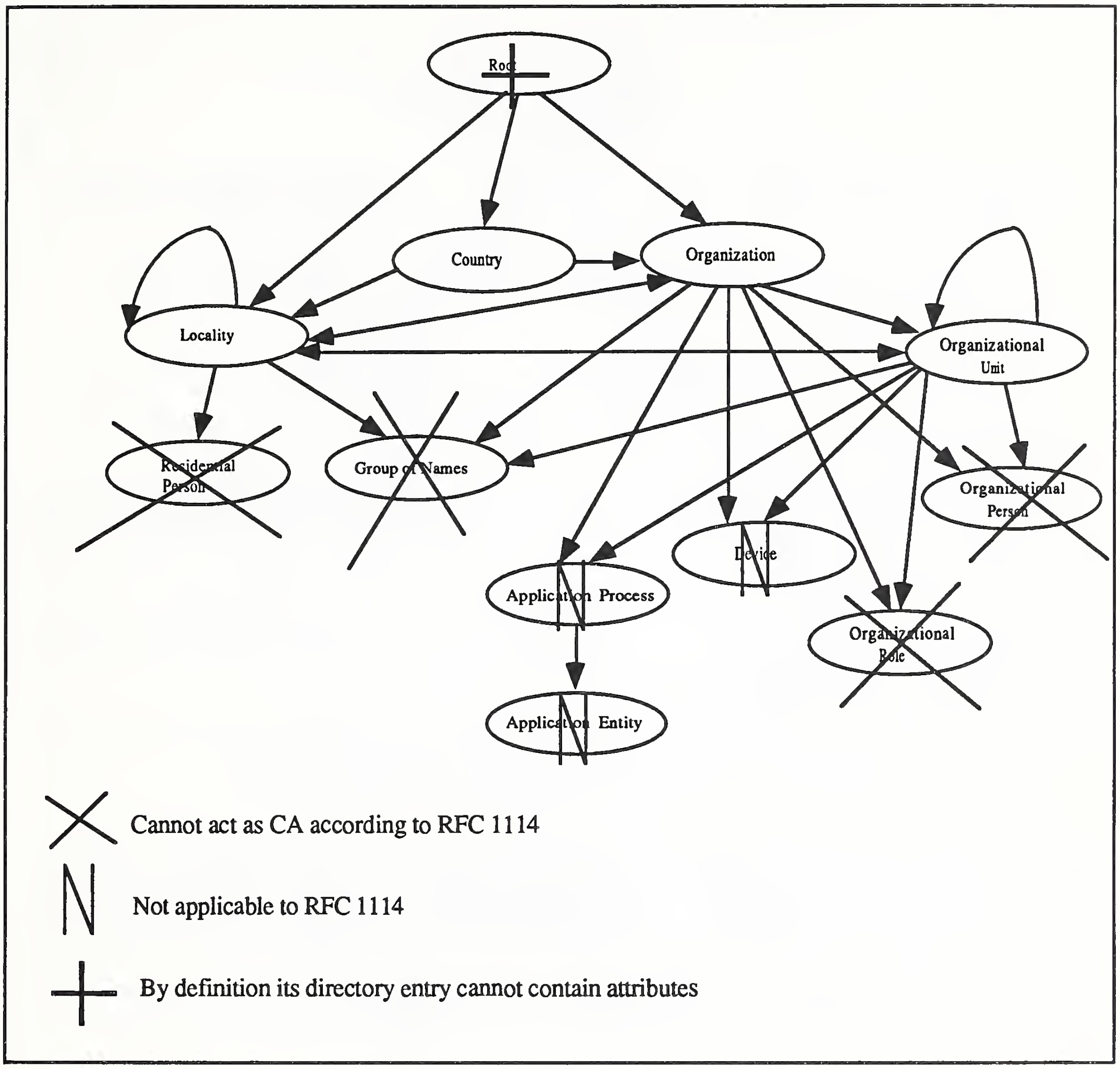


Figure 5.2.3:2 Entities in the DIT allowed to act as CAs

[RFC 1114] is designed primarily for Internet electronic mail purposes, therefore neither application processes nor application entities can be certificate holders. In the case of key management residing in the application layer, a certificate can be issued to either an application process or an application entity.

Relating the certification hierarchy to the the naming hierarchy guarantees that a certification path can always be found by direct look-up the Directory entries associated with the distinguished name of the users. As mentioned previously, this is not necessarily true if the general scheme of [ISO 9594-8] is followed. The following example illustrates what is involved in looking up the Directory entries associated with the users distinguished names. The figure below illustrates a portion of a hypothetical DIT.

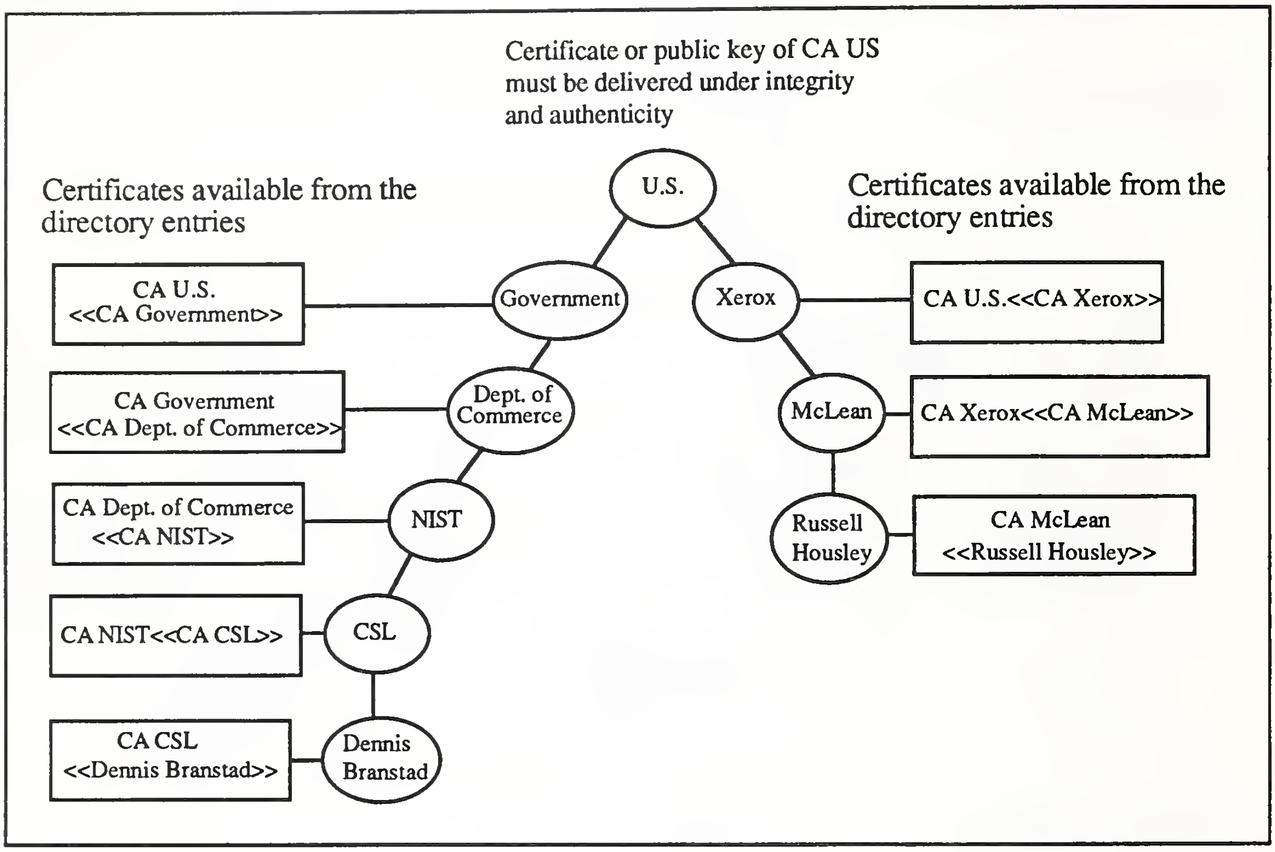

Certificate or public key of CA US must be delivered under integrity and authenticity
Certificates available from the directory entries
Certificates available from the directory entries
U.S.
Government
Xerox
Dept. of Commerce
McLean
NIST
Russell Housley
CSL
Dennis Branstad
CA U.S. <<CA Government>>
CA Government <<CA Dept. of Commerce>>
CA Dept. of Commerce <<CA NIST>>
CA NIST<<CA CSL>>
CA CSL <<Dennis Branstad>>
CA U.S.<<CA Xerox>>
CA Xerox<<CA McLean>>
CA McLean <<Russell Housley>>

Figure 5.2.3:3 Hypothetical hierarchy where Dennis and Russ are leaf nodes

Each Directory entry higher in the hierarchy than the leaf entries of users Dennis and Russ, is associated with a relative distinguished name. In the example, the entity associated with a relative distinguished name (e.g., an organization (Xerox), organizational unit (CSL), locality (McLean) or country (U.S.)) also acts as a CA. As we will see later this is not a necessity. The distinguished name of the users are:

{c = U.S., o = Government, ou = Department of Commerce, ou = NIST, ou = CSL, cn = Dennis Branstad}

and

{c = U.S., o = Xerox, l = McLean, cn = Russell Housley}.

This example assumes that Dennis is in possession of his own certificate. By accessing the Directory entries of CSL, NIST, Department of Commerce and Government; Dennis can obtain the certificates that are needed for Russ to validate Dennis′ certificate. Therefore it is possible for Dennis to send a signed message together with a certification path to Russ. The certificates provided by Dennis makes it possible for Russ to verify the signature.

The scheme of [RFC 1114] only makes use of forward certificates except at country level. An entity represented in the DIT and allowed to act as CA, for example, an organizational unit, is not required to act as CA. The figure below illustrates a CA hierarchy where CSL is not acting as CA, but where NIST does. NIST issues a certificate for user Dennis, where CSL is part of the distinguished name appearing in the subject part of the certificate of Dennis.

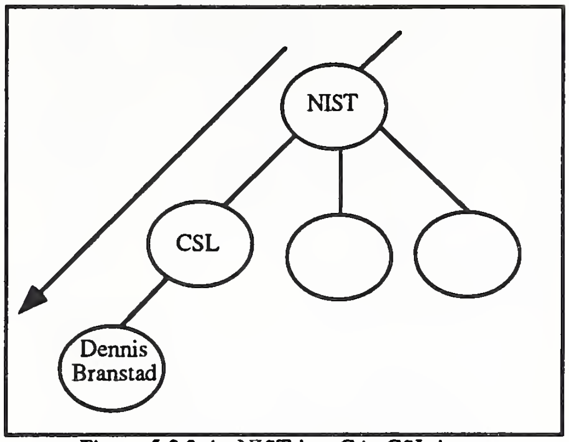


Figure 5.2.3:4 NIST is a CA, CSL is not

In the figure below a different CA hierarchy is shown, where CA NIST issues a CA certificate for CSL. CA CSL in turn issues a certificate for user Dennis. In this case CA CSL is allowed by CA NIST to issue certificates in its name space:

{c = U.S., o = Government, ou = NIST, ou = CSL}.

Thus, the name space is: (U.S.).Government.NIST.CSL.xxxx.

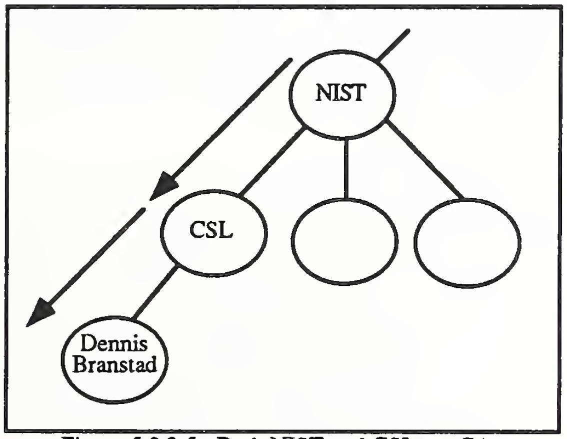


Figure 5.2.3:5 Both NIST and CSL are CAs

If the scheme of [RFC 1114] is followed, the user (if the terminology of [RFC 1114] is used it would be the User Agent) can create the certification path. This is illustrated in the following examples.

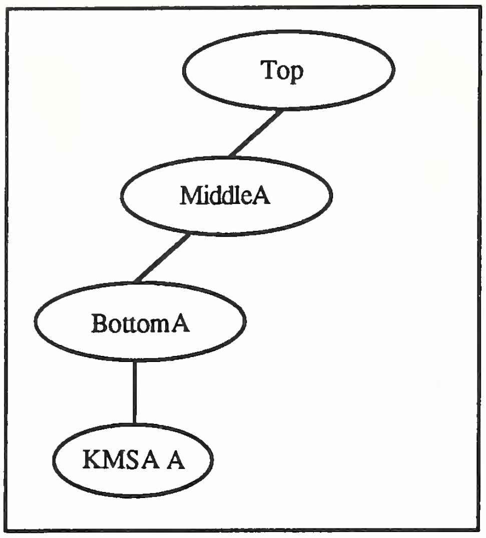


Figure 5.2.3:6 Structure of CAs above KMSA A

As a prerequisite, the user in the above figure, KMSA A, must be in possession of the public key of the topmost CA, Top. The public key of Top must be delivered under integrity and authenticity[4]. In addition KMSA A must be in possession of all certificates included in the path from Top to itself. That is, the own certificate and the CA certificates of BottomA and MiddleA. Those certificates can be obtained from the Directory. With possession of the public key of Top, it is possible for KMSA A to check the correctness of the CA certificate of MiddleA. Possessing the public key of CA MiddleA, it is possible to check the correctness of the CA certificate of BottomA, etc. Thus, it is the public key of Top that makes it possible to check the chain of certificates.

In the figure shown below KMSA A and KMSA B share the same certification authority, BottomA.

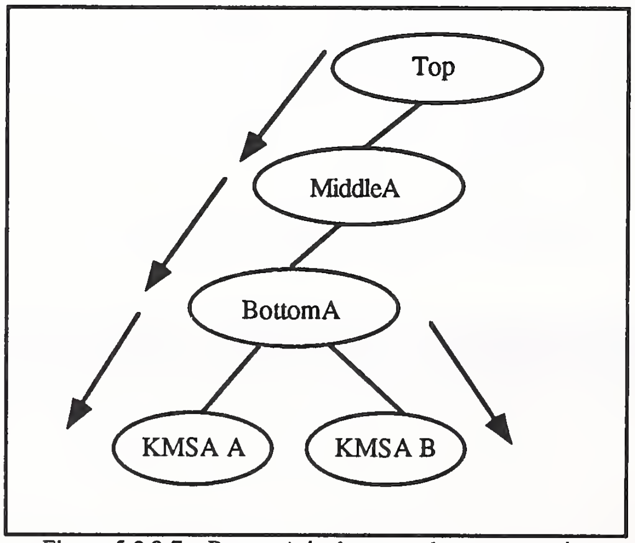


Figure 5.2.3:7 BottomA is the trusted common point

[4] Therefore it cannot be obtained through the Directory.

If KMSA B has to be able to verify a digital signature computed by KMSA A, according to the scheme of [RFC 1114] it is enough for KMSA A to send just its own certificate to KMSA B. From the distinguished names, KMSA B discovers that the common point of trust is BottomA. KMSA B which is in possession of the public key of BottomA can, thus, verify the certificate of KMSA A and thereby obtaining the public key to verify the signature.

In the figure shown below, KMSA A and KMSA B have the topmost CA, Top, as the common point of trust.

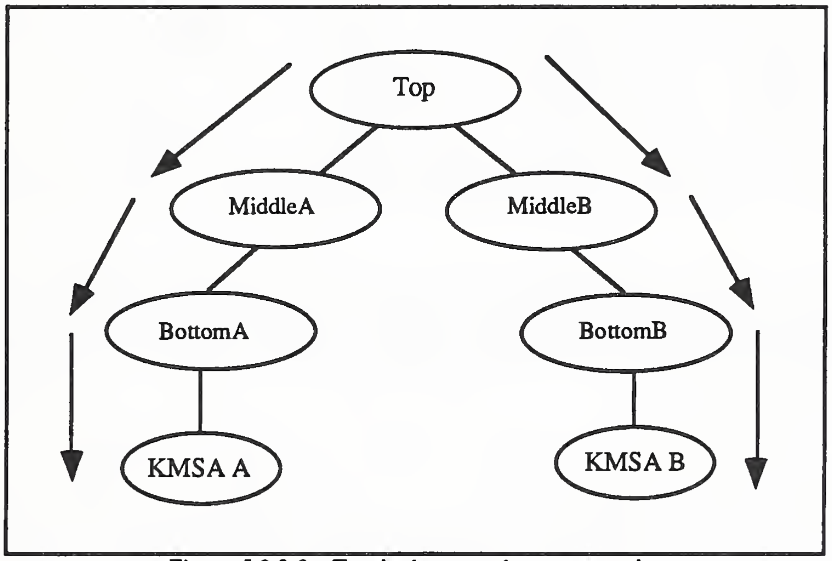


Figure 5.2.3:8 Top is the trusted common point

KMSA A has to provide the CA certificate of MiddleA, the CA certificate of BottomA and the own certificate to KMSA B, in order to make it possible for KMSA B to verify a digital signature computed by KMSA A. KMSA B, which is in possession of the public key of Top, can check the correctness of the CA certificate of MiddleA. If the certificate is valid, KMSA B extracts the public key from the certificate. With the public key of the CA MiddleA, KMSA B can check the correctness of the CA certificate of BottomA. The last step for KMSA B is to check the correctness of the certificate of KMSA A. Thereafter KMSA B can extract the public key from the certificate and verify the digital signature. If all KMSAs in the network cache the certificates included in the path between the particular KMSA and the root position, Top, they always are able to create a certification path. Thereby the time to create a certification path decreases, because fewer Directory accesses are necessary to create the certification path.

An alternative to the previously described approach is not to cache any certificates and do the Directory look-up on an as needed basis. According to Figure 5.2.3:8, when KMSA B has to verify a signature calculated by KMSA A, KMSA B can obtain the certification path by looking-up the Directory entries of BottomA and MiddleA[5]. Another possibility is that KMSA A does the Directory look-up every time it needs the certificates of BottomA and MiddleA and provides them, together with its own certificate, to KMSA B.

[5] Provided KMSA A sends its own certificate.

[RFC 1114] only allows cross-certificates between two CAs located in root position. The reason for this restriction is to avoid multiple certification paths. If in the figure below a cross-certificate is allowed between AC and BA, two possible certification paths could be created. One path is A-B-BA and the other is (A)-AC-BA.

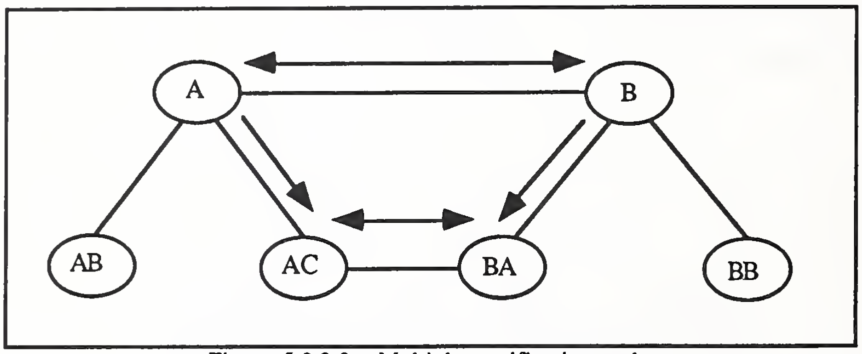


Figure 5.2.3:9 Multiple certification paths

[RFC 1114] allows only one cross-certificate to be part of a certification path. In the example illustrated in the figure below, users within the certificate domain of the U.S. cannot authenticate users within the certificate domain of Sweden. This because two cross-certificates would be included in the certification path.

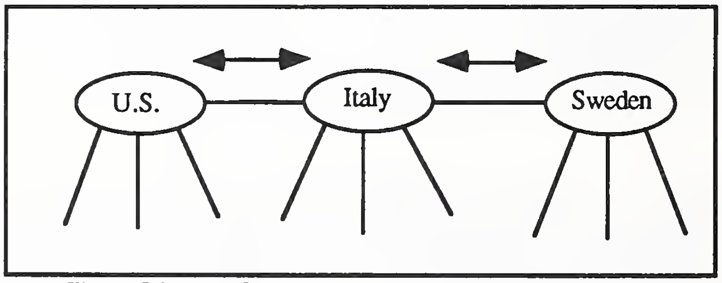


Figure 5.2.3:10 Disallowed structure of cross-certificates

To allow users within the certificate domains of the U.S. and Sweden to authenticate each other, the cross-certificates must be arranged as shown below:

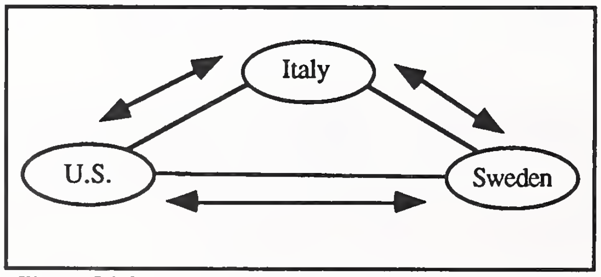


Figure 5.2.3:11 Allowed structure of cross-certificates

The reason is that agreements made between top level CAs have to be bilateral. By traversing more than one cross-certificate is therefore a violation of the bilateral rule.

5.2.4 Section summary

Of the different schemes reviewed, the Directory Authentication Framework and RFC 1114 are the ones which are most similar to our model of key management. SDNS has a single level hierarchy, which may be a limitation in an OSI environment. Is the general scheme of the Directory Authentication Framework or the restricted scheme of RFC 1114 to be preferred? The first difference between the two schemes is that the RFC 1114 scheme contains only forward certificates. From which certification authority should the trust originate? In the general scheme of the Directory Authentication Framework, see section 5.2.2, where there are both forward and reverse certificates, the trust originates from the lowest CA in the hierarchy. That is, the CA which has certified the user. With possession of the public key of the trusted CA, the user can verify the reverse certificate issued by the trusted CA and obtain the public key of the CA immediate superior in order of hierarchy. In such a scheme it may be natural for the user to trust the CA which issued the user´s certificate and other nearby CAs. In the article The Role Of Naming In Secure Distributed Systems [GASS90] it is written:

> This model matches well the notion that organizations are more likely to trust their local managers to certify their users than they would trust a higher level manager. Under this model the highest level or "root" authority will rarely be trusted for anything except for authentication spanning the highest level domains (e.g., "countries" in X.500).

In the [RFC 1114] scheme trust originates from the topmost CA, because by being in possession of the public key of the topmost CA all certificates can be verified. The second difference is that RFC 1114 presumes the certification hierarchy to coincide with the naming hierarchy. This paper does not review the Digital Distributed System Security Architecture, the reader is referred to The Digital Distributed System Security Architecture [GASS89] and Practical Authentication For Distributed Computing [LINN90]. It should, however, be noted that the Digital Distributed System Security Architecture also has a coupling between the naming hierarchy and the certification hierarchy. Very likely it is harder to implement the functionality described in the Directory Authentication Framework compared to that of RFC 1114. On the other hand, probably not everyone would agree about having the certification hierarchy parallel the naming hierarchy. Nevertheless, in this paper preference is given to the RFC 1114 styled scheme.

Note that the previous sections do not deal with trusted relationships. In reality a KMSA must have prior knowledge about the CAs it can trust. Only certificates issued by the trusted CAs can be included in a certification path.

5.3 ORGANIZATION OF KEY DISTRIBUTION AND KEY TRANSLATION CENTERS

Key Distribution Centers (CKDs) and Key Translation Centers (CKTs) are associated with a symmetric form of key management. However, early articles when describing key management based on asymmetric principles used the term Key Distribution Center. For example, the term Key Distribution Center appears both in Issues In The Design Of A Key Distribution Center [PRIC81] and Initialization Of Cryptographic Variables In A Network With A Large Number Of Computers [MATY86] although asymmetric aspects are discussed. Later articles normally use one of the following terms when describing an asymmetric form of key management: key management center, certification authority, certification center or key certification center. A user, which in our key management model is manifested by a KMSA, is referred to as a party in [ANSI X9.17] and in [ANSI X9.28] as a subscriber. The abbreviations CKD and CKT are in accordance with [ANSI X9.17].

The motive for using CKDs and CKTs is the large number of keys that otherwise have to be shared in a network constituted of n components, $O(n^2)$. Instead of letting each party share a pairwise key with every other party, they share a Key Encrypting Key (KEK) with a CKD or CKT. In [ANSI X9 17] and [ANSI X9.28] the KEK is called *KK, (*)KK or KK. *KK denotes a key pair; (*)KK denotes that the KEK can be either a key pair or a single key; KK denotes a single key. See Annex A for more details regarding single keys and key pairs. A CKD or CKT in this kind of environment is a centralized trusted party.

Annex A makes a comparison between a symmetric and asymmetric form of key management, where the asymmetric form is preferred. Why then consider a symmetric form of key management? There are several answers, organizations may have invested significantly in symmetric based key management. The investments must be paid before a transition to an asymmetric form of key management can take place. Organizations may have the opinion that symmetric algorithms are preferable to asymmetric ones. This may be particularly valid for organizations which develop their own secret algorithms. In some countries asymmetric algorithms are patented (e.g., U.S.A.) which make their usage more restricted.

The difference between a CKD and CKT environment is that in a CKD environment the CKD is responsible for the key generation. The CKD is responsible for generating the data keys (KDs) shared between two parties. Data keys can be viewed to correspond to traffic encryption keys. If [ANSI X9.17] and [ANSI X9.28] terminology is used, a data key is either an encryption data key or an authentication data key. An encryption data key may be associated with an initialization vector (IV). In a CKT environment the parties generate the keys, which can be either of type (*)KK or KD, but the keys must be notarized (translated) by the CKT. There are two possible reasons why a CKD should generate the data keys. First, a party may not have a key generation capability. Second, since the CKD is a trusted place, a party may place more trust in a key generated by the CKD than that generated by another party.

An issue is how the key centers (CKDs or CKTs) should be organized. Should there be only one key center or several? These questions are similar to the ones raised about certification authorities in the asymmetric form of key management.

The following sections use the terms offset encryption and notarization. There is more about offset encryption and notarization in Annex A.

5.3.1 Organization of Key Distribution and Key Translation Centers in X9.17

The X9.17 standard [ANSI X9.17] specifies only one key center[6], which can be either a CKD or a CKT. Thus, all the parties share the same CKD or CKT. The messages sent between a party and a key center, or between two parties, are called Cryptographic Service Messages (CSMs). CSMs can be viewed to correspond to the PDUs of OSI.

[6] However, Annex G of X9,17 describes dual CKTs.

[ANSI X9.17] specifies three kinds of environments:

- Key Distribution Center (CKD) environment;
- Key Translation Center (CKT) environment;
- Point-to-Point (PTP) environment.

The kind of keys distributed vary in the different environments. For a CKD environment the distributed keys can only be of type data keys (KDs). A CKT can translate both KDs and (*)KKs. In a PTP environment both KDs and (*)KKs are allowed to be automatically distributed. In a CKT environment when a (*)KK is present in a message it has one accompanying KD. While in a PTP environment if a (*)KK is present in a message it can have two accompanying KDs. In a CKT environment if no (*)KK is present, the message can contain two KDs. A KD may be associated with an initialization vector (IV), however, in a message there can be only one IV.

A party and a key center must share a manually distributed *KK, while in a PTP environment two communicating parties share a (*)KK. This (*)KK may or may not be manually distributed. When a (*)KK is present in a CSM, the KD present in the CSM is protected by the (*)KK. That is, the (*)KK included in the CSM is used to encrypt the KD also included in the CSM. The (*)KK must be protected with a manually distributed (*)KK. An IV may or may not be encrypted, if it is encrypted the key used to perform the encryption is the associated KD. If two KDs are present in a message, the second one is used to encrypt the IV. The key hierarchy is shown in the following figure:

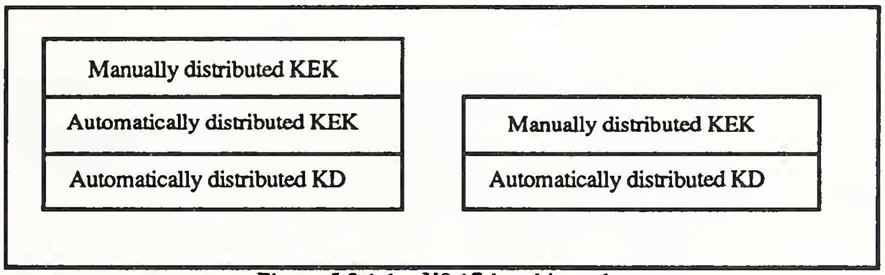

Manually distributed KEK
Automatically distributed KEK
Automatically distributed KD
Manually distributed KEK
Automatically distributed KD

Figure 5.3.1:1 X9.17 key hierarchy

In a three layer hierarchy, a manually distributed KEK is used to protect automatically distributed KEKs. Automatically distributed KEKs are then used to protect the KDs. For a two layer hierarchy a manually distributed KEK is used to protect KDs. An example of a three layer hierarchy is when Party A sends a (*)KK to be translated by its CKT. During the distribution the (*)KK is protected by a manually distributed *KK shared between Party A and the CKT. After the notarization (*)KK is sent from the CKT to Party A and thereafter from Party A to Party B. After (*)KK has been distributed to Party B, it is used to protect KDs transferred between Party A and Party B. When only one KD is included in a CSM the Message Authentication Code (MAC), which integrity protects the CSM, is calculated by using the single KD as the encryption key to produce the MAC. When two KDs are included the MAC is computed by using by a key which is derived from an exclusive-or operation performed on those KDs. A counter (CT) is used to protect against replay attacks.

The following examples describe how X9.17 key exchanges may occur. The figure below illustrates a key exchange involving a CKD:

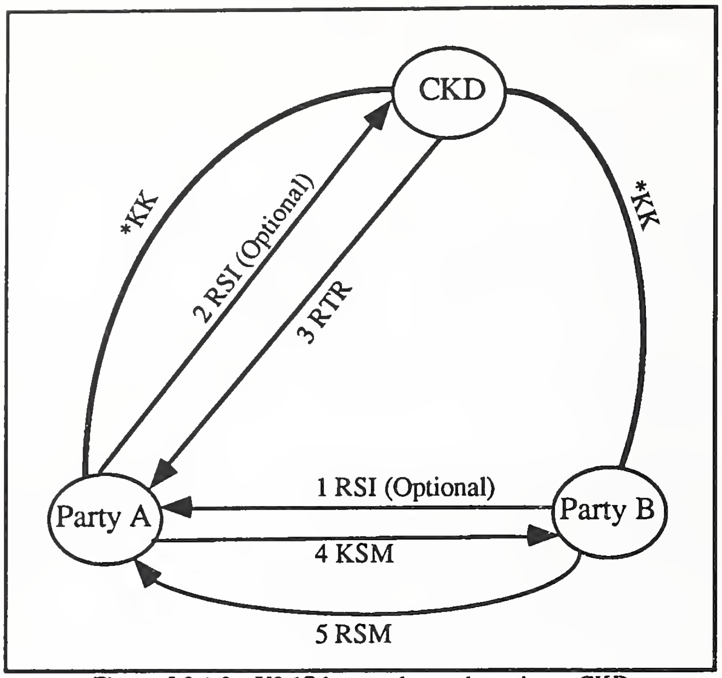


Figure 5.3.1:2 X9.17 key exchange by using a CKD

1. In this example Party B sends an RSI[7] to Party A requesting a data key (KD) and an initialization vector (IV).
2. Party A sends an RSI to the CKD requesting a data key and an initialization vector.
3. The CKD sends an RTR containing the generated key and the initialization vector, to Party A. There are two sets of keys, one set is notarized using the *KK shared between the CKD and Party A (that is, the *KK is one of the components used to produce the notarization key) and the other is notarized using the *KK shared between the CKD and Party B. The RTR CSM is integrity protected by a MAC computed with the KD. Included in the RTR is also a counter in order to detect replay attacks. In the following the presence of a counter is omitted.
4. Party A sends a KSM containing the KD to Party B. The KD is notarized using the *KK shared between the CKD and Party B. The KSM message is integrity protected in the same manner as explained in 3 above.
5. Party B acknowledges the correct receipt of the KSM message by returning an RSM. The RSM is integrity protected with the previously distributed KD.

Annex A provides a more detailed explanation about an X9.17 key exchange.

In the following two examples it is omitted that the CSMs may be integrity protected. The following diagram illustrates an X9.17 key exchange involving a CKT:

[7] RSI is a Cryptographic Service Message.

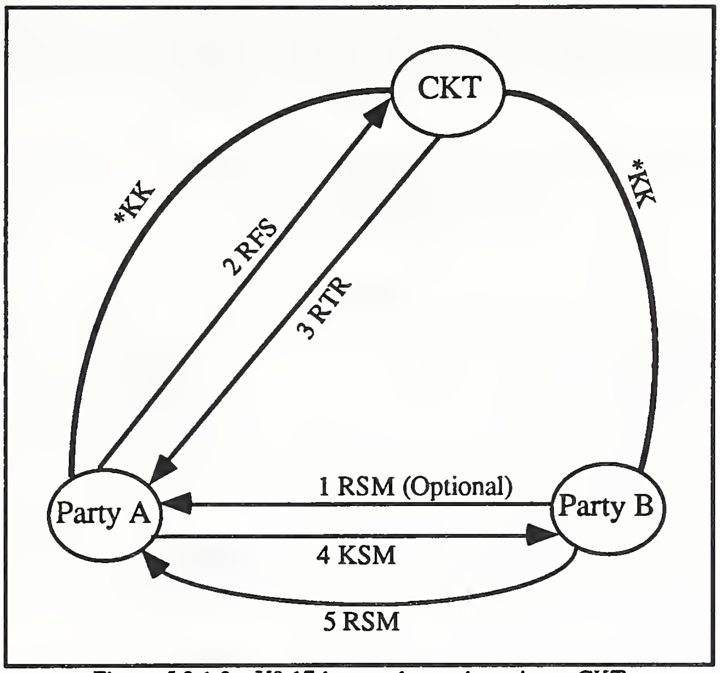


Figure 5.3.1:3 X9.17 key exchange by using a CKT

1. Party B sends a request to Party A to generate a data key (KD) and an initialization vector (IV).
2. Party A generates the KD and IV and sends an RFS to the CKT. The RFS contains the generated key material. The newly generated data key is encrypted with the *KK shared between Party A and the CKT.
3. The KD is decrypted using the *KK shared between Party A and the CKT. Thereafter the KD is notarized using the *KK shared between Party B and the CKT. The notarized KD is included in the RTR sent to Party A.
4. Party A sends a KSM to Party B, the CSM contains the notarized KD.
5. The KSM is acknowledged by an RSM.

One of the reasons why a key center is not sending the generated data keys directly to Party B is to minimize the communication with the key center. The key center, which in the case of a CKD has to be involved in every key exchange occurring in the network, may otherwise become a bottleneck.

If key notarization is not done, it is possible for a party having a keying relationship with the center, to masquerade as another party which also has a keying relationship with the center. For example, Party C could masquerade as Party A, when performing a key exchange with Party B. How such an attack could occur is described in Annex A.

In our model of key management the CKD or the CKT functionality is represented by a KCA, as previously mentioned the two parties are represented by the KMSAs.

[ANSI X9.17] provides a PTP environment. In this environment the two parties share a Key Encrypting Key, (*)KK, and need not involve a key center[8]. An [ANSI X9.17] PTP key exchange occurs in the following way:

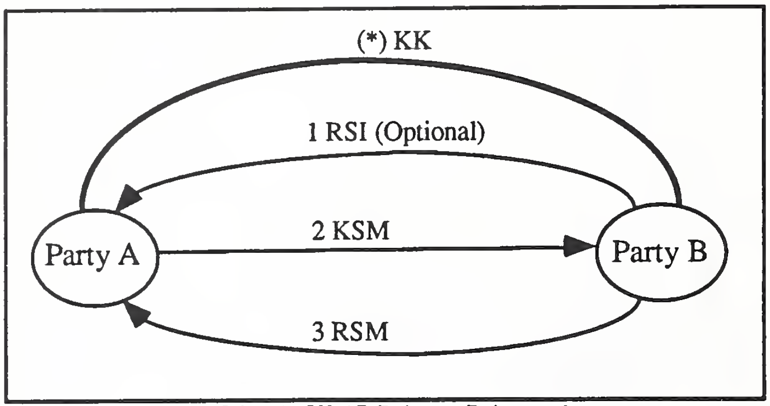


Figure 5.3.1:4 X9.17 Point-to-Point exchange

1. Party B sends an RSI requesting Party A to generate a KD and an IV.
2. Party A generates the requested key material and includes it in the KSM where the KD is protected by the (*)KK. Note, the KD does not need to be notarized.
3. Party B as an acknowledgement returns the RSM.

5.3.2 Organization of Key Distribution and Key Translation Centers in X9.28

This standard, X9.28 [ANSI X9.28], includes several key centers. As in the case of having just one CA, it seems unrealistic that just one key center (CKD or CKT) can serve a large OSI community.

A multiple center group (also called group) is a set of two or more centers which have formally agreed to share a common identity and work together to provide cryptographic keying services to their subscribers. To the subscribers of a group, the group functions as if it were a single X9.17 key center that is known by a common identity. A specific key center may belong to more than one group. The following is an example of an [ANSI X9.28] multiple center group.

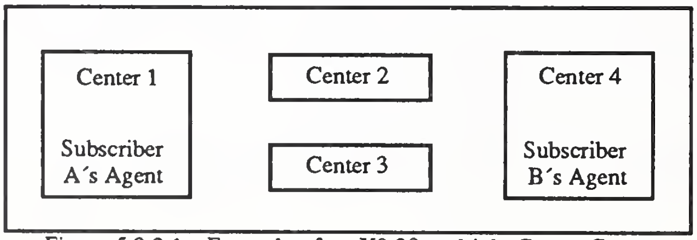


Figure 5.3.2:1 Example of an X9.28 multiple Center Group

[8] However, a CKT may have been involved when the (*)KK was established.

To its subscribers a group offers either a key distribution service, equivalent to the service provided by an X9.17 Key Distribution Center, or a key translation service equivalent to the service provided by an X9.17 Key Translation Center.

The figure below is an example of how a multiple center group may be used in an internet environment. Every LAN has a designated key center, KDC, and two KDCs could form a multiple center group.

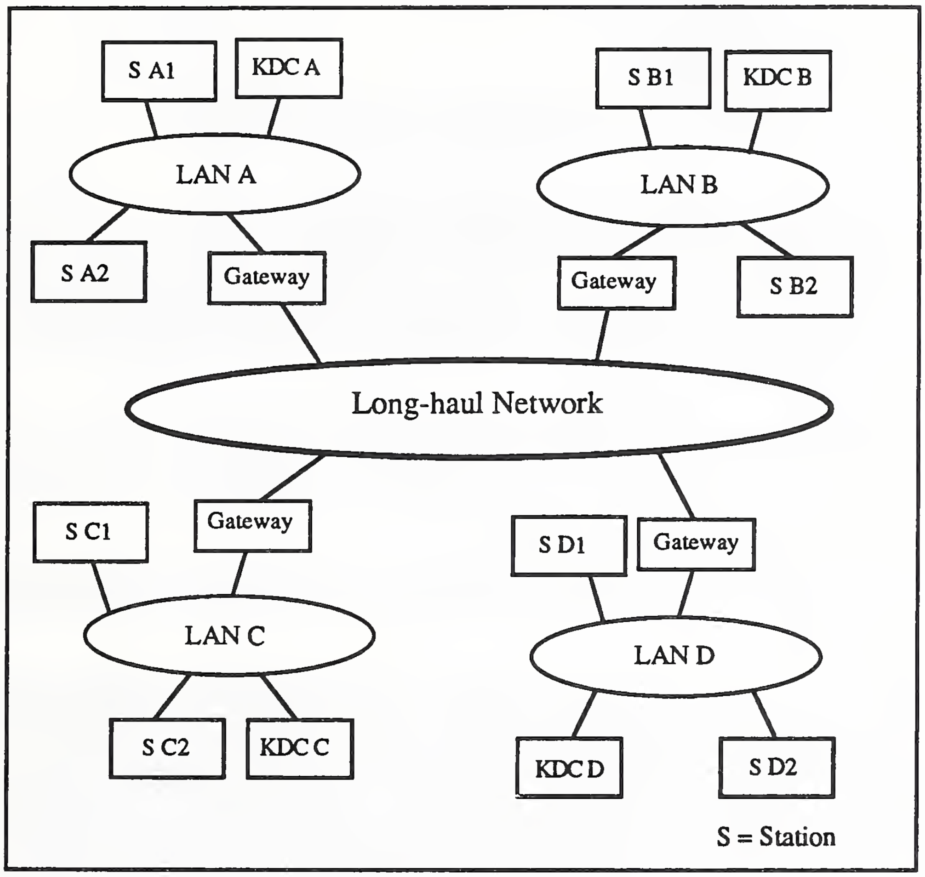


Figure 5.3.2:2 X9.28 used in an internet environment

If stations A1 and A2 exchange keys, they use their common KDC, KDC A. If stations A1 and B1 have to exchange keys they do not have a common KDC. Therefore KDC A and KDC B may form a multiple center group. Thus, enabling a station residing in LAN A to perform a key exchange with a station residing in LAN B.

A group which provides a key distribution service has centers which generate, translate and notarize keys. At least one of the centers within the group must have a key generation capability. The following figure shows an example where only one of the centers, Center 4, has the key generation capability.

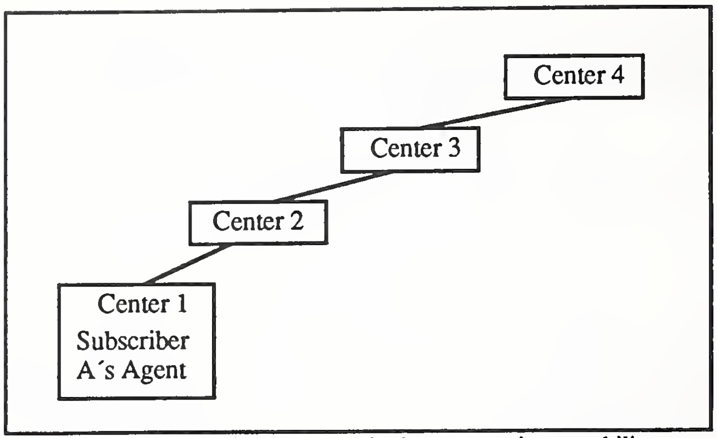


Figure 5.3.2:3 Center 4 has the key generation capability

Because Center 1 does not have a key generation capability, a request from Subscriber A is forwarded to the next center, Center 2. Center 2 also lacks a key generation capability and therefore forwards the request. Sooner or later the center which has the key generation capability is reached, in this example Center 4. If several centers have a key generation capability, during an [ANSI X9.28] key exchange transaction, only one of the centers shall use the capability. Only the subscriber´s agent shall notarize the keys.

An agent shares a key encrypting key, *KK, with each of its subscribers. The key is distributed according to [ANSI X9.17] key center rules. Two centers having a direct keying relationship share:

- a manually distributed Key Encrypting Key, *KK;
- an authentication data key (KDA) distributed manually or by using [ANSI X9.17] point-to-point protocols.

The key encrypting keys and data keys carried in multiple centers CSMs and delivered to subscribers for their use are called *subscriber* keys. The keys used by key centers for the secure transportation of subscriber keys are called *transportation* keys. When a subscriber key, (*)KK, is present in a message, the (*)KK is the highest level subscriber key transported in the message. When no subscriber (*)KK is present in a message, the subscriber KDs are the highest level keys in the message. A pair of centers in a group with a direct keying relationship share at least one *transportation key encrypting key pair*, *KK, and at least one *transportation key authentication data key*, KDA. Transportation *KKs are used to offset encrypt the highest level subscriber keys. Transportation KDAs are used to authenticate (according to OSI terminology it would be integrity protect) the CSMs exchanged between centers. Centers may use automatically distributed *KKs, distributed using [ANSI X9.17] point-to-point protocol procedures. During the distribution phase the distributed *KKs are protected with the manual distributed *KK.

In the example shown below there are two manually distributed keys, *KK(1-2) and *KK(1-3), shared between two key centers. Those keys are used to automatically distribute transportation key encrypting key pairs and transportation authentication data keys using [ANSI X9.17] point-to-point protocol procedures. For example, *KK(1-2) is used to protect the transportation key encrypting key pair and the transportation authentication key data key shared between Center 1 and Center 2 during the distribution phase.

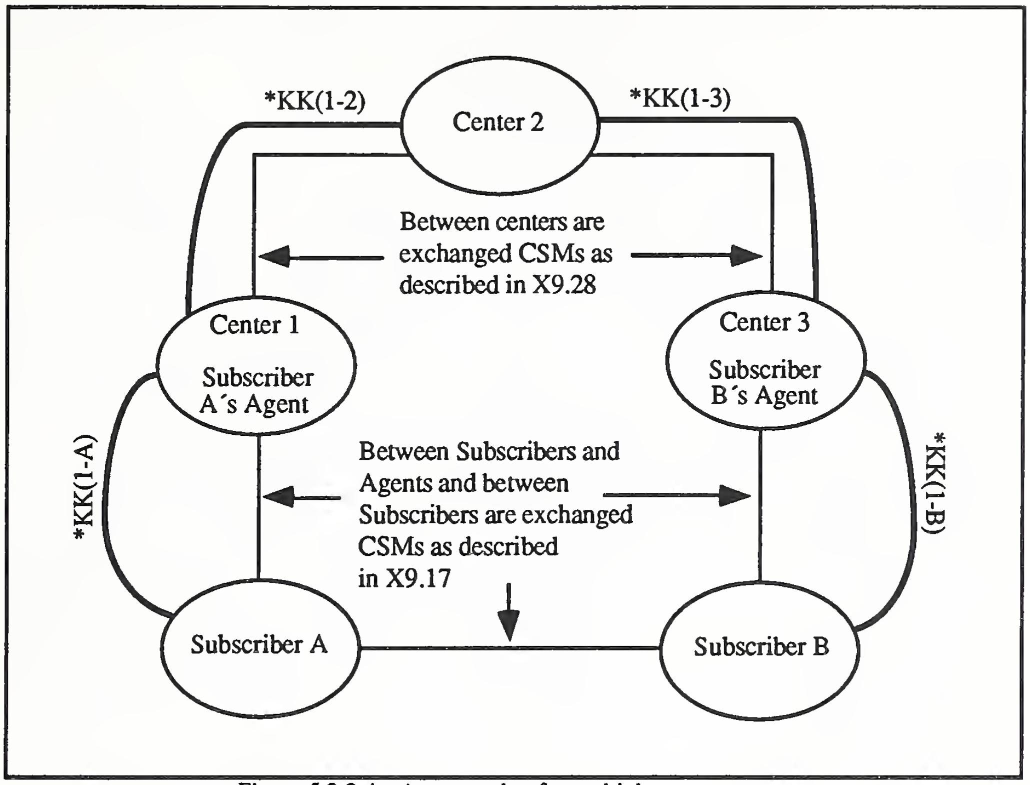

*KK(1-2)
Center 2
*KK(1-3)
Between centers are exchanged CSMs as described in X9.28
Center 1
Subscriber A´s Agent
Center 3
Subscriber B´s Agent
*KK(1-A)
*KK(1-B)
Between Subscribers and Agents and between Subscribers are exchanged CSMs as described in X9.17
Subscriber A
Subscriber B

Figure 5.3.2:4 An example of a multiple center group

As an alternative the transportation key encrypting key pairs and transportation authentication data keys can be distributed manually.

5.3.2.1 X9.28 key exchange transaction

The following example closely follows a similar example in Discussions Of Symmetrical (Secret Key) and Asymmetrical (Public Key) Cryptography [BARK91]. Center C2 is responsible for generating the keys. It is assumed that the transportation key encrypting key pairs $*KK_{A,C1}$, $*KK_{C1,C2}$, $*KK_{C2,C3}$, $*KK_{C1,C3}$ and $*KK_{B,C3}$ and the transportation key authentication data key pairs $*KDA_{C1,C2}$, $KDA_{C1,C3}$ and $*KDA_{C2,C3}$ are manually distributed.

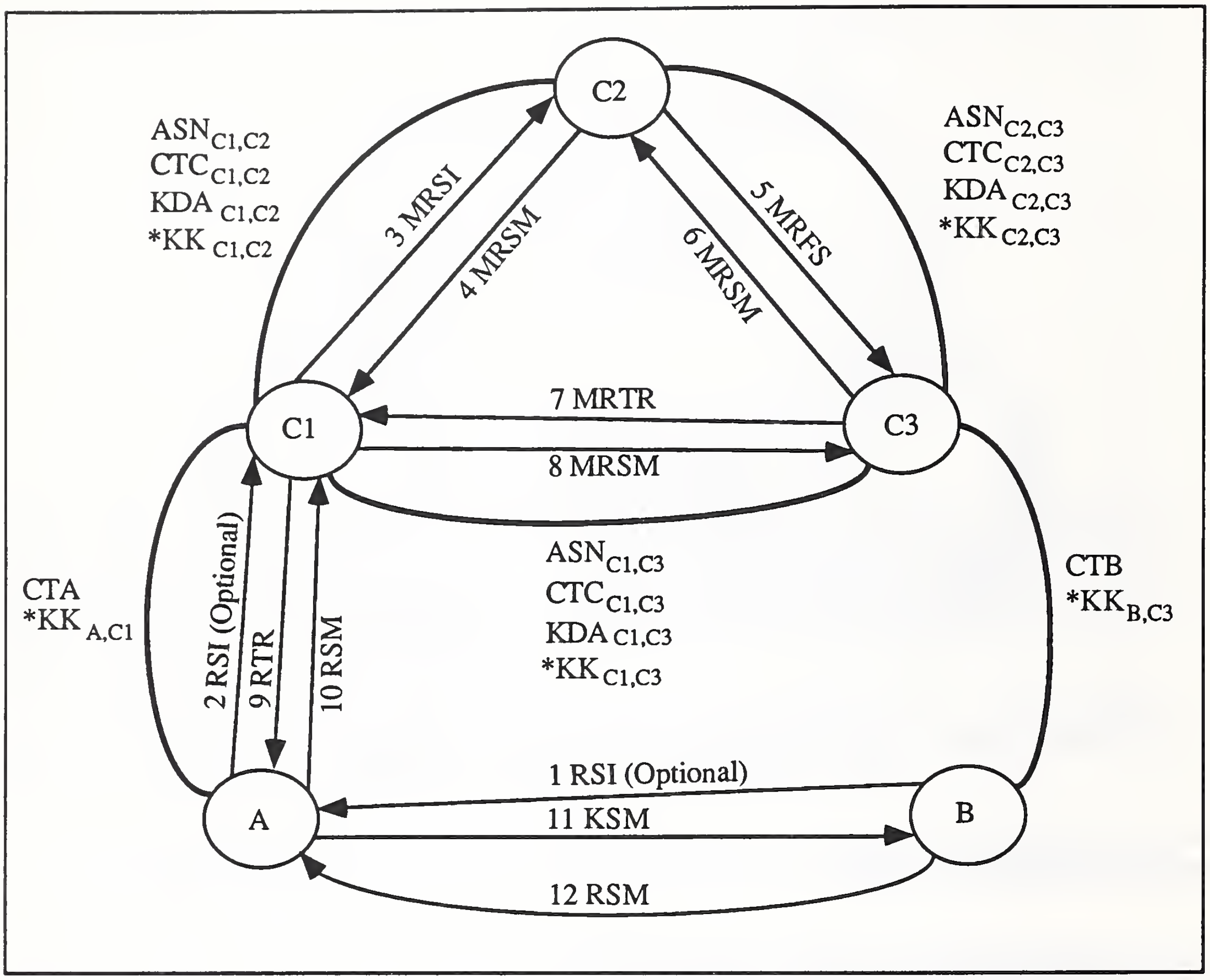


Figure 5.3.2.1:1 X9.28 key exchange transaction

- Subscriber B sends an RSI message to Subscriber A requesting the desired key material. In this example the requested keying material consists of an encryption data key (KD) and an initialization vector (IV).
- Subscriber A forwards the request to its Agent (C1).
- Upon receiving the [ANSI X9.17] RSI message from Subscriber A, C1 sends the request on to C2 in the form of an [ANSI X9.28] MRSI message, which is similar to the RSI message. The MRSI message is integrity protected by a MAC produced with $KDA_{C1,C2}$, and is replay protected by the Authentication Sequence Number $ASN_{C1,C2}$. Each [ANSI X9.28] message is protected by a MAC produced with a KDA and is replay protected by an ASN. In the following description this is omitted.

- C2 acknowledges the correct receipt of the MRSI message by returning an MRSM message. C2 generates the requested keying material. The *KK and the counter shared between C2 and C3, $*KK_{C2,C3}$ and $CTC_{C2,C3}$, are used to produce the offset encryption key. The offset encryption key is used to encrypt the highest level subscriber key. In this example the highest level subscriber key is the KD. If desired, the IV is encrypted with the KD. The offset encrypted KD and the IV are included in the MRFS. C3 acknowledges the correct receipt of the MRFS message by returning an MRSM
- C3, which is Subscriber B´s Agent, offset decrypts the KD, if necessary decrypts the IV, and notarizes a copy of the KD. (The KEK and counter shared between Party B and its Agent, that is, $*KK_{B,C3}$, and CTB, are used to produce the notarization key.) This copy is intended for Subscriber B. Another copy of the KD is offset encrypted by using $*KK_{C1,C3}$ and $CTC_{C1,C3}$. The two copies are part of the MRTR message sent to C1. C1 acknowledges the correct receipt of the MRTR message by returning an MRSM.
- C1 offset decrypts Subscriber A´s copy of the KD and notarizes it. Nothing is done to the copy intended for Subscriber B. The notarized copy intended for Subscriber A and the notarized copy intended for Subscriber B are both placed in the [ANSI X9.17] RTR message which is sent to Subscriber A. The RTR message is integrity protected by a MAC produced with the KD, as explained in [ANSI X9.17]. Subscriber A acknowledges the correct receipt of the RTR message by returning an RSM.
- Subscriber A sends Subscriber B´s copy of the keying material in a KSM message.
- Subscriber B responds with an RSM if the KSM is received correctly.

Note that during an X9.28 key exchange transaction the key material may be visible at the different key centers. A feeling may be that this is not good, because a secret should be shared by as few as possible.

<u>5.3.2.2 Diversion from the key management model</u>

Our model of key management represents the key centers by KCAs whose communication stream follows a single route. The KCAs of the key management model have a hierarchical relationship, however, [ANSI X9.28] centers do not need to be hierarchically related. For example, if a multiple center group consists of two key centers, each of them attached to a subscriber, the centers are not related in a hierarchical manner. In the hierarchy, the message flow between KCAs is restricted to a single path. However, for [ANSI X9.28] the intermediate centers used in one route of the message flow may or may not be the same as those used in the return route of the message flow.

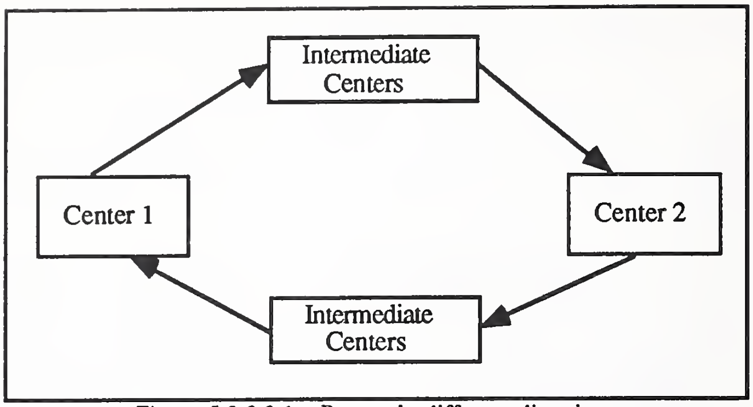


Figure 5.3.2.2:1 Routes in different directions

The key management model does not include the possibility of message flows going in different routes between KCAs. The possibility of having the message flow going in different routes may be more difficult to achieve in an OSI application protocol. Before an application layer PDU can be sent, an application association must be created[9], through the use of ACSE. ACSE is also used to terminate the application association. If we try to adopt application layer concepts on Figure 5.3.2.1:1, should the application association be terminated or remain open after the MRSM is sent from C2 to C1? It may be more efficient to use already opened application associations, that is, between C1-C2 and C2-C3, instead of creating a new one, that is, between C1-C3. Application protocols include a state machine and therefore the PDUs must be sent and received in a certain order. If OSI application layer concepts are used, it may be more natural to relate the MRTR sent from C3 to an existing association, the one created between C2-C3, instead of sending the MRTR in a newly created application association.

5.3.3 Section summary

In an [ANSI X9.17] environment there is just one key center. As for certification authorities it seems unrealistic that a single key center can serve a large OSI community spread across organizational as well as national boundaries. An [ANSI X9.28] environment which includes several key centers is more suitable for an OSI environment. However, this is contrasted by a weakness of potentially having the key material visible at the different centers during an [ANSI X9.28] key exchange transaction.

5.4 PASSIVE OR ACTIVE CERTIFICATION AUTHORITY

Should the CA take on a passive or an active role? In an active role approach a CA can produce certificates on-line. If the on-line functionality approach is required, the private key of a CA must be permanently stored in a system. If our model has to be realized and it has to be possible to perform rekeying (see section 5.6 for definition of rekeying) it is a prerequisite to have on-line functionality, because the new certificates have to be signed by the KCA on-line. If on-line functionality is not offered, the rekeying must be done by manual means or by staged rekeying. Staged rekeying means that a KMSA delivers its new public key to the KCA, but the KCA issues the new certificate at a later point in time. A KMSA cannot in this case create a protected channel to the KCA when the public key is delivered. But, the KCA can produce the certificate off-line and

9 Only connection oriented application protocols are considered.

the KMSA will be able to verify the certificate when it is delivered, because the KMSA is in possession of the public key of the KCA. This is similar to the approach described in Initialization Of Cryptographic Variables In A Network With A Large Number Of Terminals [MATY86].

The cryptographic facility where the private key has to be stored is of importance. [FIPS 140-1] specifies the requirements for a cryptographic facility to be used by U.S. Government Agencies processing unclassified or unclassified but sensitive information. Four different levels are mentioned in the FIPS, where level four is the most advanced and probably the most expensive to realize. It can be inferred that a cryptographic facility where the private key of a KCA is stored, probably has to have level three or four functionality. If a level three or four cryptographic facility is too expensive to realize, maybe economical aspects also must be considered when determining whether the KCA has to take on an active or passive role. The draft FIPS Key Management Using ANSI X9.17 [FIPS-171] can be used as a guideline; it specifies that the cryptographic module must have level 3 functionality. The Certificate Signature Unit (CSU) mentioned in [RFC 1114] must also have level 3 functionality. The CSU is used to generate the public and private keys of a CA, and sign certificates and certificate revocation lists (CRLs) issued by a CA.

5.5 GENERATION OF THE PRIVATE AND PUBLIC KEY PAIR

In an asymmetric form of key management, which entity should create the private and public key pair? The certification authority or the user? The relationship between the CA and the user is important. If they are not part of the same organization, the user may not trust the CA to create the keys. From the CA´s point of view, it may even be an advantage to allow the user to create the public and private key pair. All the CA may be willing to do is to vouch for the binding between the user´s distinguished name and the public key component. The CA may not want to put itself in a situation where a user denies having signed a certain message. The user may argue that another entity has knowledge of his private key and has taken advantage of that knowledge.

In the following paragraphs the approaches taken to key generation by different standards are reviewed.

5.5.1 Generation of public key pair in SDNS

From the public documents available, that is, [SDNS 90-4262], it appears that in SDNS the Key Management Center (KMC) creates the private and public key pair. When rekeying is performed, the Rekey request PDU (Rkrq), sent from the SDNS component to the KMC, does not contain the information to be certified. However, the Rekey response PDU (Rkrs) sent from the KMC to the SDNS component contains the new credentials and key materials of the component.

5.5.2 Generation of public key pair in ISO/CD 11166

The document is focused on solutions where the user creates its own private and public key pair and sends the public part to the Certification Key Center (CKC). The CKC certifies the public key and returns a certificate to the user.

5.5.3 Generation of public key pair in the Directory standards

The Directory Authentication Framework [ISO 9594-8] does not mandate which entity should create the key pair. Both possibilities are considered; the user or the CA can be the creator.

5.5.4 Generation of public key pair in RFC 1114

It is the user that creates his or her private and public keys. A reference implementation of the RFCs included in Privacy Enhanced Mail for Internet, will contain the key generation software. The Certificate Signature Unit (CSU) mentioned previously is only needed when creating the keys of a CA.

5.6 REKEY HANDLING

Using our model of key management to explain the purpose of rekeying, it can be described as follows:

- In an asymmetric form of key management, the rekey procedure automatically distributes the new private key and the certificate, alternatively just the certificate, from a KCA to a KMSA, or from a KCA to its subordinate KCA.
- In a symmetric form of key management, the rekey procedure automatically distributes the new Key Encrypting Key (KEK) shared between two KMSAs, a KMSA and KCA, or two KCAs.

Note that the term rekeying is not formally defined in any standard document. In this paper it is bound together with the key management model. There are some differences in terminology used in this area. In documents [ISO 10736] and [ISO 11577] the replacement of a traffic encryption key when the cryptoperiod has expired, is called rekeying.

Should KCAs and KCAs be allowed to perform on-line rekeying, or should manually based methods be used? Different security domains may have different opinions about this issue. One motive not to allow rekeying is that the key material, which in this context is the private key or a KEK, has a cryptoperiod. The reason to have a cryptoperiod is that the key material cannot be valid for an indefinite period of time because there is a risk that it might become compromised. Since rekeying is usually done near the end of the cryptoperiod, the risk that the key material is compromised is at its highest.

An advantage of having the rekey functionality on-line is that manual intervention is eliminated or minimized. In a large network high costs are associated with installing the key material manually.

5.6.1 Rekey handling in an asymmetric form of key management

The following example explains how rekeying can be performed. The illustration is done by using our model of key management: a KMSA and a KCA exchanges two PDUs in order to establish the new certificate of the KMSA. In a request PDU the KMSA delivers the new public key and in a response PDU the KCA returns the certified public key in the form of a certificate. The production of a certificate according to this example will occur on-line, that is, the active role approach.

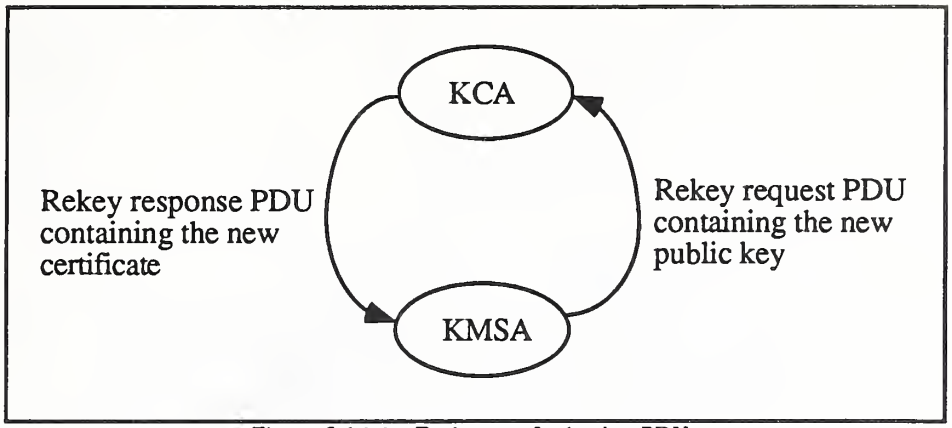

KCA
KMSA
Rekey response PDU containing the new certificate
Rekey request PDU containing the new public key

Figure 5.6.1:1 Exchange of rekeying PDUs

One important question is whether authentication alone is enough of a security measure to safely issue a certificate. For example, a KMSA proves its identity, and, thus, demonstrates to be in possession of valid (current) credentials. If the KMSA shows it has valid credentials, will its immediate superior KCA be allowed to issue the KMSA the future private key and certificate, alternatively just the certificate?

If there is only concern about integrity, rekeying could be done by combining manually and automatically based methods. The new public key is then transferred automatically from the KMSA to the KCA but also verified by manual means.

A security domain may allow the KMSAs to be rekeyed, but may not allow the KCAs to do be rekeyed. The reason for this limitation is that the private key of a CA is involved if the KCAs are rekeyed. If the private key is compromised, false certificates can be distributed. As mentioned in [RFC 1114], if the private key of a CA becomes compromised all the subordinate certificates must be revoked and new certificates generated to replace the old.

Another design issue arises when rekeying is allowed. Which entity should create the new private and public key pair? This issue has already been discussed in paragraph 5.5.

A problem which arises when new certificates are issued during rekeying is that the certificates of two KMSAs may become "out of phase". Every KMSA and KCA cannot perform the rekeying at the same time. At what point in time should the new certificate be allowed to be used? Note that a private key is always associated with a certificate. Figure 5.6.1:2 illustrates how the KCA and the KMSAs can be related in order for the "out of phase" problem to occur.

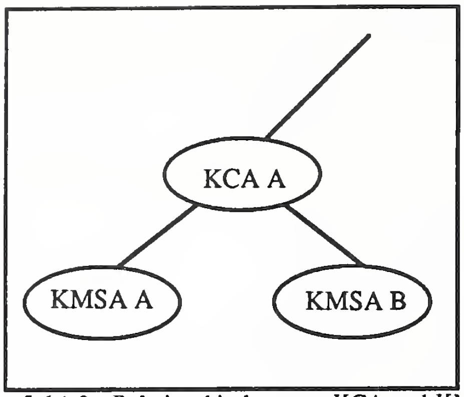


Figure 5.6.1:2 Relationship between KCA and KMSAs

At time T_1 KCA A creates its new private and public key. At time T_2 KMSA A performs the rekeying and KCA A signs the certificate with its new private key. The old certificate is then revoked. After T_2, KMSA A cannot perform a key exchange with another KMSA that has not yet rekeyed and been issued a certificate signed with the new private key of KCA A. According to Figure 5.6.1:3 between T_2 and T_3 KMSA A and KMSA B are not able to perform a key exchange.

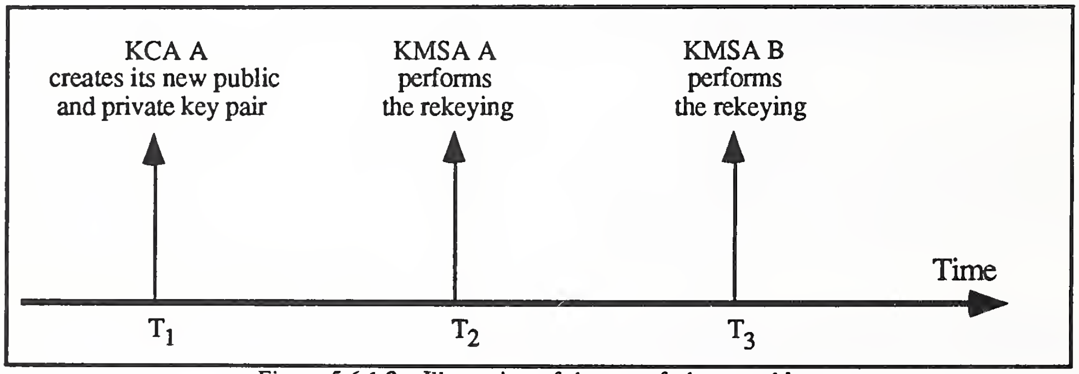

KCA A creates its new public and private key pair
KMSA A performs the rekeying
KMSA B performs the rekeying
Time
T1
T2
T3

Figure 5.6.1:3 Illustration of the out of phase problem

The reason why KMSA A and KMSA B cannot perform a key exchange is that they cannot verify each others certificates because they are signed with different private keys.

The problem may be solved in two different ways:

1. The new certificates are not valid until the expiration of the old.
2. Both the old and new certificates are valid during a limited period of time.

In both cases a KMSA is in possession of two certificates.

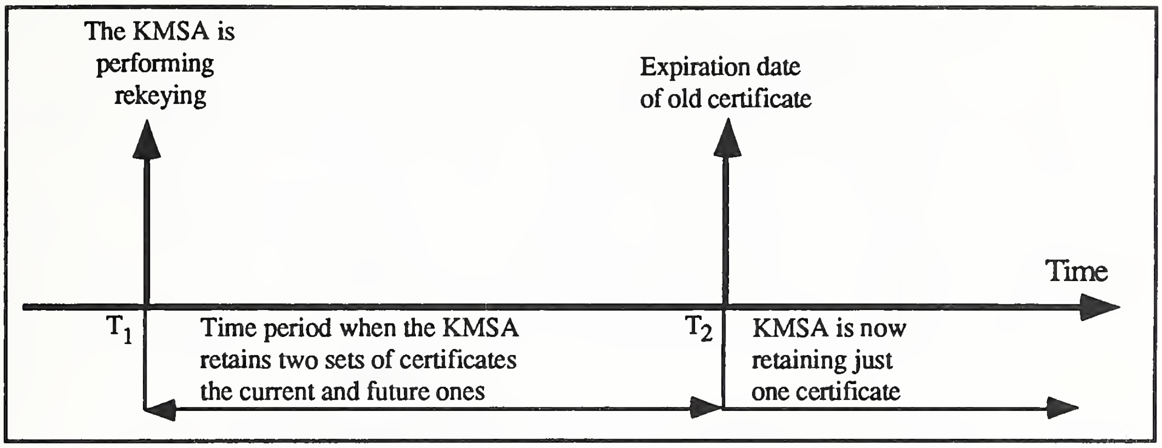

The KMSA is performing rekeying
Expiration date of old certificate
Time
T_1
Time period when the KMSA retains two sets of certificates the current and future ones
T_2
KMSA is now retaining just one certificate

Figure 5.6.1:4 The KMSA is in possession of two sets of certificates during time period T_1-T_2

According to the first alternative the new certificate will be valid when the old one expires. There is no overlapping time when both the old certificate and the new certificate are valid. Not having an overlapping time can cause some problems.

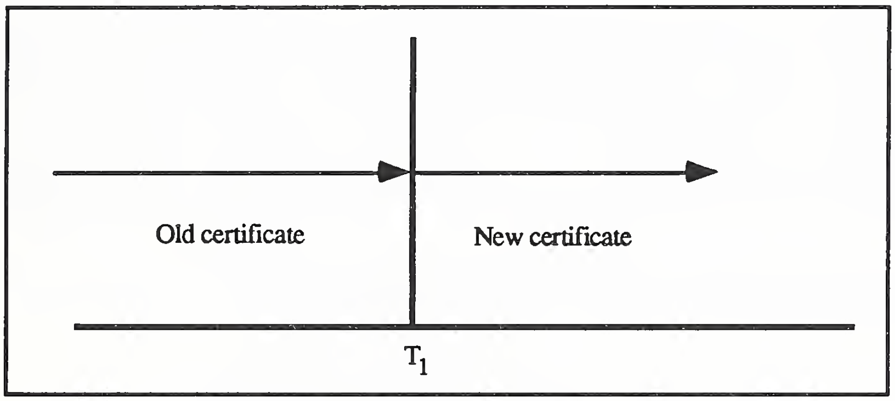
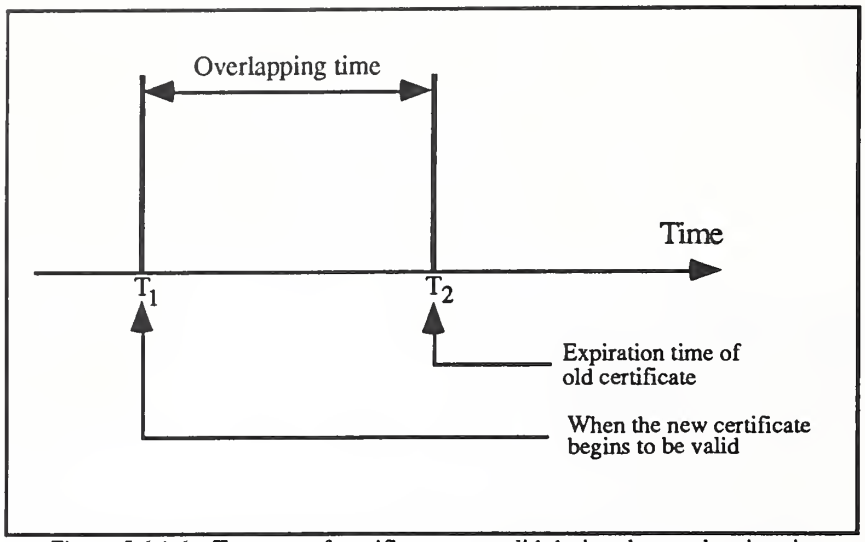


Figure 5.6.1:5 At T_1 the old certificate is substituted

There can be different opinions between two KMSAs about which of the certificates to use when they perform a key exchange. Synchronization between the two KMSAs may be required if they perform a key exchange near time T_1. If there is no synchronization, one of the KMSAs may have the opinion that the old certificate is the valid one, while the other claims that the new certificate is the valid one. Figure 5.6.1:6 shows the second alternative where two certificates are valid for an overlapping period of time.


Overlapping time
Time
T_1
T_2
Expiration time of old certificate
When the new certificate begins to be valid
Figure 5.6.1:6 Two sets of certificates are valid during the overlapping time

The old certificate may be used close to time T_1, while close to time T_2 the new certificate may be used.

If an overlapping time is allowed and the new certificates are signed with a new private key, that is, the KCA has also changed its private and public keys, a problem arises. Which of the KCA public keys should be used to verify the certificate of the peer? The old one associated with the old certificate or the new one associated with the new certificate? Can this be decided from the validity field included in the peers certificate? Another solution is to have a key identifier which identifies the private key used to sign the certificate. Such an identifier is available in the certificate structure described in [ISO 11166] but it is not available in the certificate structure of [ISO 9594-8].

When there are a lot of components involved, rekeying becomes a very complex procedure. Problems that may occur are illustrated in the following example:

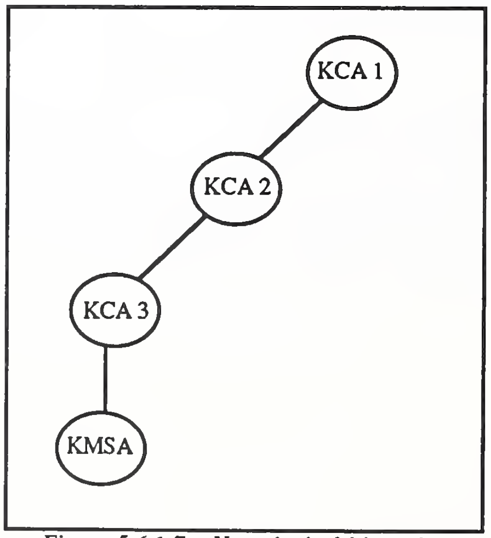


Figure 5.6.1:7 Hypothetical hierarchy

The KMSA initiates the rekeying by sending its new public key component to KCA 3. However, the certificate given to KCA 3 by KCA 2 is going to expire. Should KCA 3 be permitted to certify the new public key of the KMSA before it has gotten its own public key certified? According to the scheme in [RFC 1114] by giving KCA 3 a forward certificate, KCA 2 authorizes KCA 3 to sign certificates within its domain. When the KMSA needs to get its public key certified, KCA 3 has an authorization that is about to expire. If KCA 3 signs the certificate with the private key associated with the expiring certificate, the certificate issued to the KMSA will be revoked when the certificate of KCA 3 expires. This is because it is a practice to revoke all certificates signed with a private key that is no longer valid. Furthermore, KCA 2 could also have an authorization that is about to expire and must therefore create a new public pair, and so on. However, the complexity involved when rekeying is done in a hierarchy of CAs is an intrinsic part of key management and cannot be avoided.

A way to manage the rekeying is to schedule the KCAs according to their place in the hierarchy. The highest KCA in the hierarchy creates its new private and public key. Thereafter the KCAs which are on the second level in hierarchy create their new public key pairs. A second level KCA can then send the new public key for certification since the first level KCA has already created its new public key pair. The uppermost KCA can sign the certificate of its subordinate KCA with the new private key. Thereafter the third level KCAs can create their new public key pairs and so forth. See the following figure:

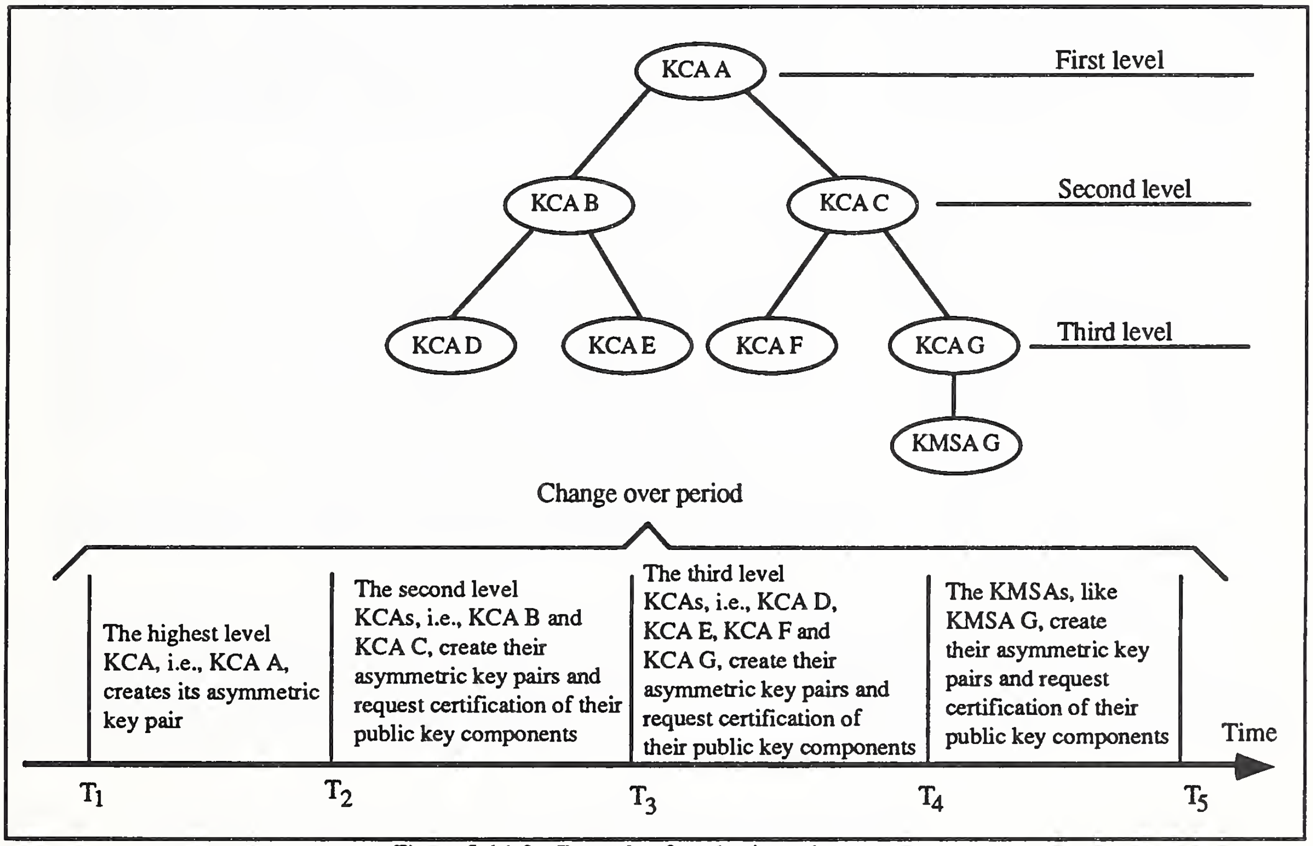

KCA A
First level
KCA B
KCA C
Second level
KCA D
KCA E
KCA F
KCA G
Third level
KMSA G
Change over period
The highest level KCA, i.e., KCA A, creates its asymmetric key pair
The second level KCAs, i.e., KCA B and KCA C, create their asymmetric key pairs and request certification of their public key components
The third level KCAs, i.e., KCA D, KCA E, KCA F and KCA G, create their asymmetric key pairs and request certification of their public key components
The KMSAs, like KMSA G, create their asymmetric key pairs and request certification of their public key components
Time
T1
T2
T3
T4
T5

Figure 5.6.1:8 Example of a rekeying scheme

When KMSA G performs rekeying, KCA C has already certified KCA G. Therefore KCA G is allowed to issue certificates in its name space by using the new private key. Note that this example illustrates a "worst case" scenario, in reality certificates of KCAs probably will have longer validity periods than certificates of KMSAs. So in most cases all components in the network need not perform the rekeying. However, it appears difficult to fully automatize the rekeying procedure described above. Probably the security officers in the different certificate domains have to agree about the different time periods and manually coordinate the change over. During the change over period the rekeyed KCAs and KMSAs are in possession of two certificates.

5.6.1.1 Rekey handling in SDNS

The SDNS key management documents [SDNS 90-4262] discuss two different types of rekeying:

- interactive rekeying;
- staged rekeying.

In both cases, the responsibility for initiating a rekey lies with the KMSA. The Rekey request (Rkrq) and Rekey response (Rkrs) PDUs are used for interactive rekeying. An SDNS component sends an Rkrq to the Key Management Center (KMC) and the KMC returns an Rkrs. For staged rekeying the Staged Rekey request (Srkrq), Staged Rekey response (Srkrs), Staged Rekey Delivery request (Srdrq) and Staged Rekey Delivery response (Srdrs) PDUs are used. When staged rekeying occurs, an SDNS component acts as a rekey agent. To the rekey agent comes a rekey request. The rekey agent after receiving the request contacts the KMC to get new credentials and key materials for the request. After the rekey agent has the new credentials and key materials, the original SDNS component can contact the rekey agent again and obtain the credentials and key materials. [SDNS 90-4262] gives no detailed explanation about the key materials and credential components.

The SDNS documents [SDNS 90-4262] do not explain in detail why there are two different forms of rekeying or when one form or the other should be used. The explanation given in [SDNS 90-4262] is that staged rekeying is a useful service when direct access to the Key Management System is unavailable. The devices may therefore trust an intermediary to collect requests and obtain new credentials and key materials. An example where staged rekeying is needed is when a mail user wants to get new credentials. The mail user may reside in an environment where transport class four is not part of the OSI stack. However, there may be a requirement to have transport class four in the key management stack. The user sends a request to the rekey agent by secure mail. The rekey agent in turn can use the key management stack.

Although it is not stated in [SDNS 90-4262], the contents of the PDUs give an indication that the KMC creates the new key materials. Thus, the public and private key pair of an SDNS component are created by the KMC.

The available SDNS documents [SDNS 90-4262] do not mention how the "out of phase" problems are handled. In informal discussions it has been said there are three time periods associated with an SDNS credential.

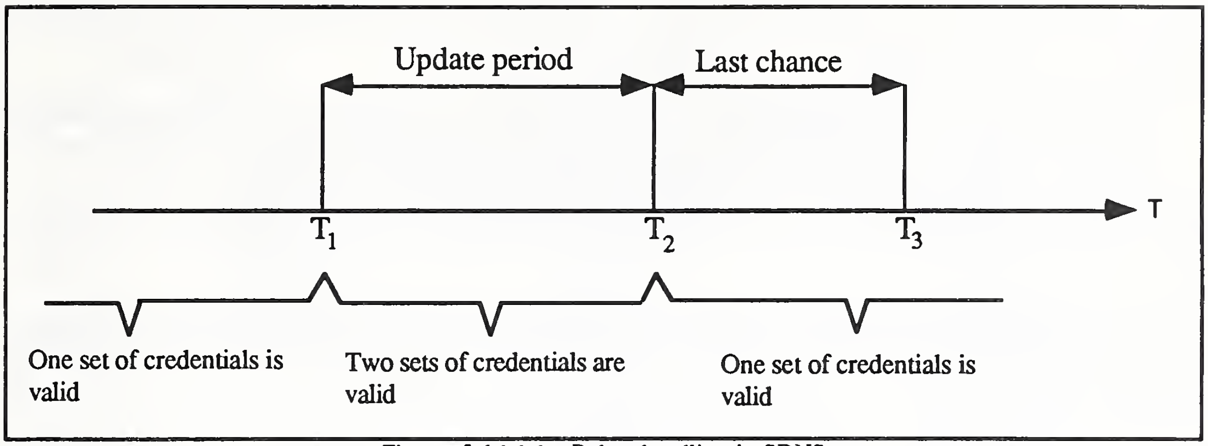

Update period
Last chance
T
T_1
T_2
T_3
One set of credentials is valid
Two sets of credentials are valid
One set of credentials is valid

Figure: 5.6.1.1:1 Rekey handling in SDNS

After time T_1, an SDNS component can obtain new credentials and key materials from the KMC. During the update period, between times T_1 and T_2, the SDNS component may be in possession of two sets of credentials and key materials. During the update period the SDNS component can still perform key exchanges with SDNS components that may or may not have performed the rekeying. If an SDNS component has not performed the rekeying by the end of the update period, it cannot perform key exchanges with other SDNS components. However, during the last chance period an SDNS component can still obtain new credentials and key materials from the KMC. If the KMC has to change its public and private key pair, it is possible to have the new public key, certificate or whatever it might be, as part of the new credentials or key materials distributed. See the definition below of the Rkrs. Having just one KMC eliminates much of the complexity described previously. [SDNS 90-4262] mentions two kind of credentials and key materials, see the below PDU description:

```
Rkrs ::= SET {
   ref-num       INTEGER OPTIONAL,
   kms-cred-A          [0] IMPLICIT Credentials,
   new-k-mat-A         [1] IMPLICIT KMaterials,
   kms-cred-B          [2] IMPLICIT Credentials,
   new-k-mat-B         [3] IMPLICIT KMaterials,
   crl-ver             Crl-Ver,
   new-crl             [4] IMPLICIT KeyList OPTIONAL,
   new-univ-crl        [5] IMPLICIT KeyList OPTIONAL,
   no-more             [6] IMPLICIT BOOLEAN OPTIONAL,
   ekr           BOOLEAN }
```

Nothing is mentioned in [SDNS 90-4262] about whether some manual verification is employed or needed as a complement to the rekeying.

The above description of SDNS rekeying probably illustrates only the tip of the iceberg. Presumably, there are a lot of details which have not been made public.

5.6.1.2 Rekey handling in the Directory standards

The Directory Authentication Framework [ISO 9594-8] states that the certificate production should occur off-line and must not be performed with an automatic query/response mechanism.

This definition precludes at least interactive rekeying. [ISO 9594-8] gives the following explanation:

> The advantage of this certification is that because the secret[10] key of the certification authority, CA, is never known except in the isolated and secure CA, the CA secret key may then only be learnt by an attack on CA itself, making compromise unlikely.

[ISO 9594-8] mentions that when a new certificate is installed it may for some time overlap with the old.

The above phrase does not appear to prohibit a staged form of rekeying. When staged rekeying is employed, the certificate is delivered with an automatic query/response but the production can occur off-line.

The following two sections investigate the possibility to perform rekeying when the Directory is involved. Previously, it was mentioned that the Directory standards prohibit an interactive form of rekeying. If there is a requirement to perform an interactive form of rekeying and the Directory is involved, there are several problems. The following section attempts to list the problems. A prerequisite for interactive rekeying is that the private key of a CA is available on-line (earlier described as taking on an active role). Rekeying can be performed according to two different schemes. The approach of the first scheme is to include the rekeying functionality in the Directory standards, which means the Directory standards must be augmented. The second scheme does not use the Directory to perform the rekeying but the result of the rekeying is stored in the Directory.

In SDNS there is a peer to peer relationship between an SDNS component and the KMC[11]. The SDNS component always knows the presentation address of the KMC and can therefore establish an application association with it. New credentials are delivered through an integrity and confidentiality protected channel, protecting the communication between the KMC and the SDNS component. The protected channel is created by an interactive form of key exchange. As we will notice, the Directory environment is different.

5.6.1.2.1 Rekeying as part of the Directory standards

The idea investigated in this section is to realize the KCA functionality by a Directory System Agent (DSA). Hopefully, it is well known that a user of the Directory accesses the Directory through the Directory User Agent (DUA).

The first problem encountered is that the available Directory services provided to the Directory user which are: read, compare, list, search, abandon, add entry, remove entry, modify entry and modify relative distinguished name do not include rekeying.

The peer to peer relationship may not exist in the Directory environment. The Directory is distributed, and users of the Directory are not aware of which Directory System Agent (DSA) holds the information. [ISO 9594-4] contains the following figure of how the DSAs can interact:

[10] The term private key is used in this paper.

[11] The interactive form of rekeying is in mind.

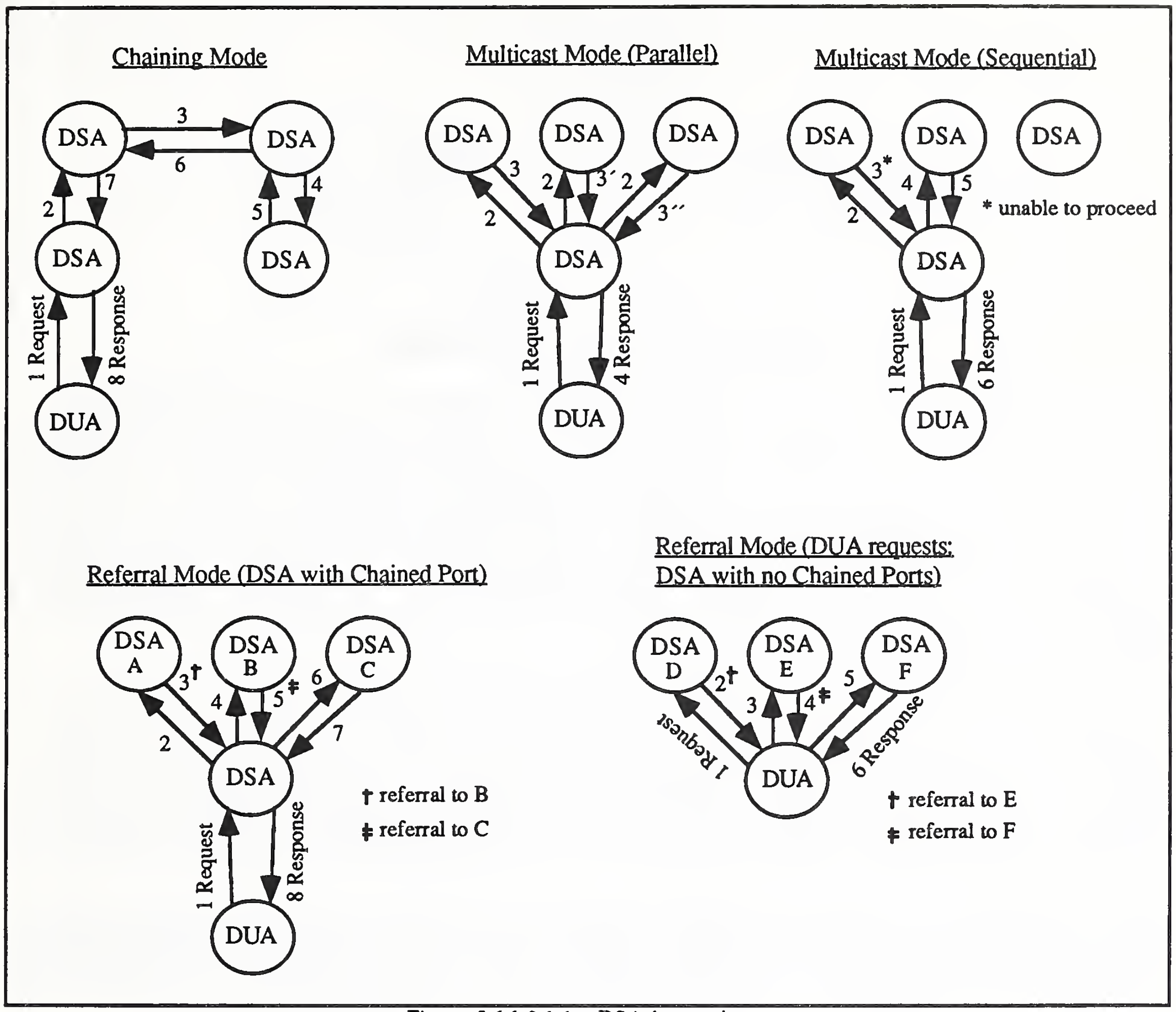


Figure 5.6.1.2.1:1 DSA interactions

The different ways of how DSAs may interact have one thing in common, the operation is always initiated by a DUA. As described in our key management model, rekeying also occurs between two KCAs. Since a KCA is manifested by a DSA, there is a need for operations originating from a DSA.

Consider the situation where rekeying is included as a Directory service. When a rekeying occurs, protected channels are created between the involved DSAs. However, some intermediate DSAs may not be trusted by either the source or destination, posing the threat that a newly created private key may be disclosed during the distribution phase. For example, when a chain operation is employed, DSA 1 which is associated with CA A, needs to perform a rekeying with DSA 3, which is associated with CA B. But, the traffic passes through DSA 2.

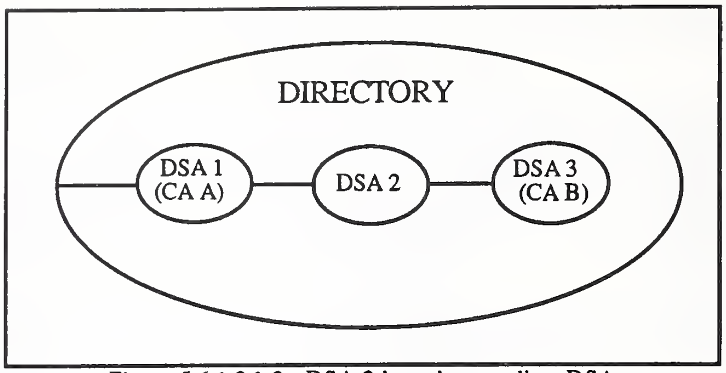


Figure 5.6.1.2.1:2 DSA 2 is an intermediate DSA

To have an intermediate DSA is not desirable when the CA creates the public and private key pair, because the private key is exposed to a third entity. However, it is possible for the user initiating the rekey operation to prohibit chaining and thereby eliminate the described drawback. In both the referral and multicast cases DSA 1 can create an application association with DSA 3. It is then assumed that authentication takes part before any rekey information is sent. Also the superior references and cross references could be used to obtain the presentation address of the remote DSA. See the article Distributed Operations Of The X.500 Directory [SMET91] for more details. Another solution is to have a store and forward based type of rekeying, the data passing intermediary systems would then not be exposed.

Since the Directory is based on the Remote Operation Service Element (ROSE), to have rekeying as part of the Directory standards forces the protocol between the KMSA and KCA and between two KCAs also to be based on ROSE. It logically follows that the protocol between two KMSAs would be based on ROSE, because it would be strange to have separate families of application protocols. A special Key Management Application Service Element for KMSA-KMSA purposes is required and protocols based on ROSE are required for KMSA-KCA and KCA-KCA purposes.

According to the Directory standards, a DSA can be the holder of a certificate, so it can perform strong authentication. It may seem strange that the DSA can be in possession of other private keys, such as the ones which are associated with CAs. It becomes unclear how to perform the authentication. If a DSA performs the authentication it is not enough, since the entity wanting a new certificate must be convinced of communicating with the right certification authority.

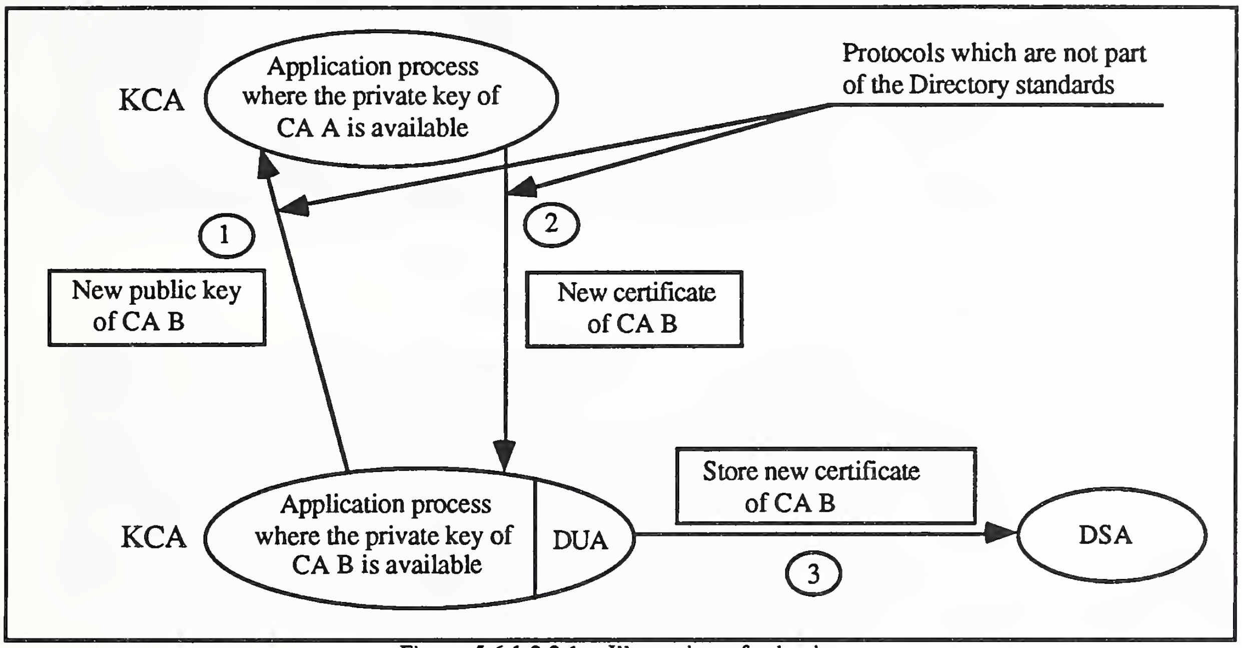
A DSA is, of course, allowed to contain certificates signed by different certification authorities. If therefore the private keys of CA A and CA B are associated with the same DSA, and CA A signs a certificate intended for CA B, rekeying becomes a local matter outside the scope of OSI.

Another issue is the level of trust that can be provided by the Directory. A Key Distribution Center or Key Management Center is normally considered as a place where a high level of security precautions, for example, physical security, are taken. It is questionable if the Directory can be viewed as such a place.

Because of the problems encountered it seems to be a good idea to keep rekeying outside of the Directory.

5.6.1.2.2 Rekeying performed outside the Directory

The rekeying scheme described in this section does not make use of the Directory to perform the rekeying, but instead is the result of the rekeying, a certificate, stored in the Directory. The KCA functionality is manifested by an application process. How rekeying is performed is illustrated in the figure below:



KCA
Application process where the private key of CA A is available
Protocols which are not part of the Directory standards
1
New public key of CA B
2
New certificate of CA B
Store new certificate of CA B
KCA
Application process where the private key of CA B is available
DUA
DSA
3

Figure 5.6.1.2.2:1 Illustration of rekeying

The KCA (manifested by the application process holding the private key of CA B) initiating the rekeying knows the presentation address of its superior KCA (manifested by the application process holding the private key of CA A).

Rekeying is initiated by letting the application process associated with CA B send the new public key for certification. The application process associated with CA A returns a certificate. The application process associated with CA B stores the certificate in the Directory. The application process is then acting as a Directory user.

5.6.1.3 Rekey handling in ISO/CD 11166

[ISO 11166] states that the public key of the Key Certification Centre [Center] (CKC) could be distributed automatically but the correctness of the key must be verified by an independent method. Note that [ISO 11166] does not use the term CA, instead the term CKC is used. [ISO 11166] describes a CKC to be: "A facility operated by the certification authority which generates and returns certificates." An independent method means that some form of manual verification must be done. A verification method is explained in the committee draft. According to this method, the CKC and the party sending the public key to the CKC should agree on a one-way function to be used. The check value computed from the one-way function must be at least 32 bits. The check value of the public key must be distributed by the CKC over multiple independent channels. Examples given in the committee draft of independent channels are surface mail and facsimile.

[ISO 11166] is oriented towards the approach where the user creates its own public and private key pair and sends the public key component to the CKC for certification. No particular method is recommended in the committee draft of how this can be achieved. The criteria that a particular method must satisfy are:

- The CKC shall verify the authentic origin of the submitted key.
- The CKC shall authenticate the submitted key value.

Annex E of [ISO 11166] contains different proposals of how the public key can be certified. One of the proposals is very similar to the method explained in the article Initialization Of Cryptographic Variables In A Network With A Large Number Of Terminals [MATY86]. [MATY86] gives a description of how a user[12] can register his or her public key at the CA the first time, that is, the user has previously not been registered at the CA. It is not clear from [ISO 11166] if the proposals are only valid for a party who wants to certify its public key the first time, or if the procedure should be repeated every time the party needs to get a new certificate.

5.6.1.4 Section summary

When there is a hierarchy of certification authorities rekeying becomes a very complicated procedure. If the Directory is involved and used as a repository for the certificates, problems due to both the Directory standards and the Directory structure are encountered. Therefore, rekeying should be performed outside the Directory. As in SDNS having only one certification authority for the whole network makes rekeying much easier to perform. However, an OSI environment is constituted by many different organizations which are spread across national boundaries, and having just one certification authority in such an environment seems unrealistic. Therefore, when a hierarchy of certification authorities are involved the complexity associated with rekeying cannot be avoided and can be regarded as an intrinsic part of key management.

5.6.2 Rekey handling in a symmetric form of key management

Since a symmetric form of key management is well known and has been used for a long time, organizations using it have established routines for how to perform the rekeying. The "out of phase" problems described for asymmetric key management cannot occur, because a Key Encrypting Key (KEK) is shared only between two parties. When the two parties have changed the KEK no further complications arise.

5.6.2.1 Rekey handling in X9.17

The following key hierarchies are allowed in [ANSI X9.17]:

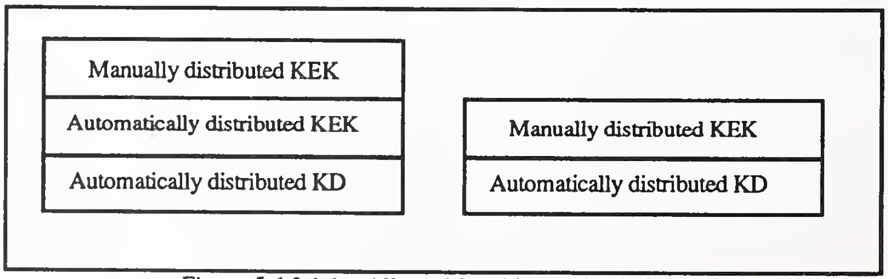

Manually distributed KEK
Automatically distributed KEK
Automatically distributed KD
Manually distributed KEK
Automatically distributed KD

Figure 5.6.2.1:1 Allowed key hierarchies in X9.17

12 [MATY86] describes how the public key of a terminal can be registered but in this context the term user is more appropriate.

The highest key in hierarchy, the uppermost KEK, is allowed to be distributed only by manual means. The motive for this decision is that manual means are regarded safer than automatic. However, the second level KEK may be distributed automatically. Appendix B of [ANSI X9.17] gives an example of manual key distribution.

5.6.2.2 Rekey handling in X9.28

The following key hierarchies are allowed in [ANSI X9.28]:

Level 1	Manually Distributed Transportation *KK				
	between centers			between agent & ultimate recipient	between centers
Level 2	Auto Distr. subscriber KD using X9.28	Auto Distr. subscriber (*)KK using X9.28	Auto Distr. transport. *KK using X9.17	Auto Distr. subscriber. (*) KK using X9.28	Auto Distr. transport *KK using X9.17
Level 3		Auto Distr. subscriber KD using X9.28	Auto Distr. subscriber KD using X9.28	Auto Distr. subscriber KD using X9.28	Auto Distr. subscriber (*) KK using X9.28
Level 4					Auto Distr. subscriber KD using X9.28

Figure 5.6.2.2:1 Allowed key hierarchies in X9.28

As for [ANSI X9.17] the highest key in hierarchy, the level 1 key, must be distributed by manual means.

5.7 REVOCATION LIST HANDLING

The Certificate Revocation List (CRL) conveys the identities of invalidated certificates. To include every revoked certificate in its entirety would be a waste of space. Instead, the CRL contains the identifiers, for example, the serial numbers, of the revoked certificates.

The motives for having a revocation list include:

- the private key of an entity has been compromised;
- the user whose identity the certificate indicates, has left the organization which issued the certificate;
- a user may become untrustworthy and is excluded from the system.

The first two motives are also mentioned in the article Privacy For Darpa Internet Mail [LINN89].

How should the CRL be organized and distributed? Those two issues concern only the asymmetric form of key management. The following paragraphs review how different standards have solved the problem.

5.7.1 Revocation list handling in SDNS

A description of SDNS revocation list handling is given in Architectural Model Of The SDNS Key Management Protocol [LAMB88]. Two methods of updating the CRL of an SDNS component are identified. First, SDNS devices may infrequently poll the KMC and obtain the latest version of the Certification Revocation List. Second, when two SDNS devices perform a key exchange, they must also inform each other about the version number of the CRL in their possession. The device with an old version of the CRL can obtain the newer version from the other device. This approach allows for a limited number of devices to get the latest version of the CRL and then spread it through the network.

The SDNS form of certificate revocation list handling is not directly applicable to the full generality of the key management model described previously, because in SDNS there can only be a single level CA hierarchy. A realization of our model would have several CAs, and allow the KMSAs performing a key exchange to be certified by different CAs. If an SDNS form of revocation list handling is used, the problem is how to share a common CRL between several CAs. This is especially difficult to achieve when the CAs which certified the two communicating KMSAs belong to different security domains.

5.7.2 Revocation list handling in the Directory standards

The Directory Authentication Framework [ISO 9594-8] states that each CA shall maintain:

1. time-stamped list of the certificates it issued which have been revoked;
2. time-stamped list of revoked certificates of all CAs known to the CA, certified by the CA.

The following example illustrates which certificates to check for appearance in the CRL when Russ authenticates Dennis:

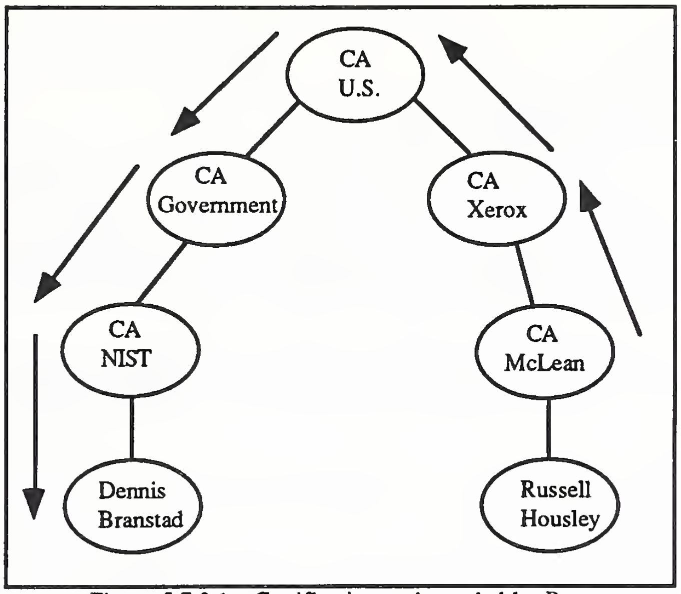


Figure 5.7.2:1 Certification path needed by Russ

The certification path needed by Russ is:

CA McLean <<CA Xerox>>, CA Xerox<<CA U.S.>>, CA U.S. <<CA Government>>, CA Government <<CA NIST>>, CA NIST<< Dennis Branstad>>

Russ must make five Directory accesses to get the associated CRLs for verification:

CRL of CA McLean to check whether CA Xerox is present;
CRL of CA Xerox to check whether CA U.S. is present;
CRL of CA U.S. to check whether CA Government is present;
CRL of CA Government to check whether CA NIST is present;
CRL of CA NIST to check whether Dennis is present.

In the example it is assumed that there are both forward and reverse certificates.

5.7.3 Revocation list handling in RFC 1114

[RFC 1114] makes some improvements to revocation list handling in [ISO 9594-8]. A [ISO 9594-8] CRL does not include information telling whether it is the latest version. If the scheme of [ISO 9594-8] is followed, when performing an authentication a KMSA must access the Directory in order to be sure it has the latest CRL. If the CA does not update the CRL often, frequent access of the Directory is both unnecessary and time consuming. What is desirable from the KMSA's point of view is to cache the CRLs associated with other KMSAs with which it frequently communicates. It could, for instance, be the CRL of its local CA and of CAs in positions close to its local CA. [RFC 1114] contains an additional field in the CRL for the purpose of identifying the next planned version of the CRL.

```
RevokedCertificateList ::= SIGNED SEQUENCE {
        signature     AlgorithmIdentifier,
        issuer        Name,
        list          SEQUENCE OF RCLEntry,
        lastUpdate          UTCTime,
        nextUpdate          UTCTime}
```

```
RCLEntry ::= SEQUENCE {
        subject CertificateSerialNumber,
        revocationDate UTCTime}
```

It should be noted that a CA may issue a new CRL before the date indicated by *nextUpdate*, however, it is guaranteed that at *nextUpdate* a new CRL is issued. If no info is added to a CRL on or before the next issue date, the same CRL is reissued. By comparing the current data/time with the *nextUpdate*, it can easily be determined whether or not the CRL is the latest version. Thereby it can be determined if a CRL included in the cache should be replaced.

Another improvement is that [RFC 1114] CRL signatures are computed over all serial numbers, while in [ISO 9594-8] every serial number is signed and then the whole CRL is signed.

5.7.4 Revocation list handling in ISO/CD 11166

[ISO 11166] mentions CRL handling in an annex, so CRL handling is not part of the standard itself. Revocation of certificates is done by letting the CA broadcast to the users the identities of the certificates revoked. The drawbacks of the solution if we try to adopt it to our model of key management are:

- When a KCA has to notify the KMSAs belonging to its jurisdiction that a certain certificate has been revoked, some of the KMSAs may not be up and running. Precautions should be taken to assure they receive the notification. If resending has to occur the protocol will probably become very complex and will be similar to the updating of a distributed database.
- KMSAs under the jurisdiction of another KCA must also be notified. This is a prerequisite, if KMSAs under the jurisdiction of different KCAs shall be able to perform a key exchange.
- The method only describes how to revoke KMSA certificates. If we describe [ISO 11166] with our general key management model, it has only one level of hierarchy. Thus, there are no descriptions of how to revoke KCA certificates.

Maybe some of these questions will be answered when the Multiple Certification Center annex is added to the standard.

The benefit of the above described method is that it makes it possible for KMSAs to be notified immediately when a certificate is revoked. The methods described previously do not provide the possibility to notify a KMSA immediately.

5.7.5 Section summary

Of the above described methods for doing certification revocation list handling, the Directory Authentication Framework [ISO 9594-8] and [RFC 1114] seem to be the ones most suitable for our key management model. [SDNS 90-4262] and [ISO 11166] are not appropriate because there are no directions given of how to handle the revocation lists when there are several certification authorities.

5.8 CONVERSION PROBLEMS

If a security domain chooses a symmetric form of key management and the other an asymmetric, is it possible for them to interact? Is it possible to design an application layer gateway that would do the needed conversions?

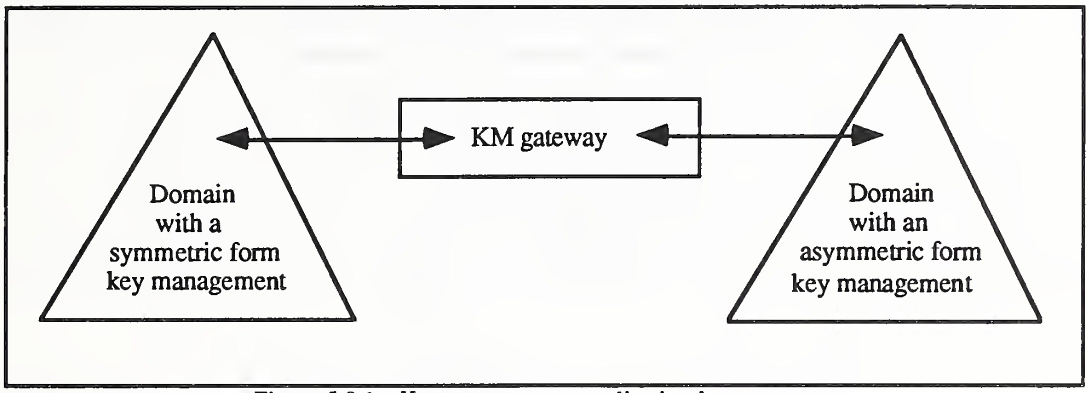

KM gateway
Domain with a symmetric form key management
Domain with an asymmetric form key management

Figure 5.8:1 Key management application layer gateway

The design of an application layer gateway that would perform the needed conversions is an issue not addressed by this paper.

5.9 PROTECTION OF GROUP AND BROADCAST KEYS

A non-centralized form of key management, like the certificate-based form, has no central place accessible to all users. How can distribution of group and broadcast keys occur when there is no central place? For instance, the SILS SDE protocol [IEEE P802.10B] needs keys shared by several entities for group communication and broadcasting purposes. Another example of a need for group keys is a distributed environment where multi-peer communication is possible. If there is a central place, for example, a Key Distribution Center, which is accessible for every station in a LAN, the problem can be solved. The article Encapsulation Security Protocol Design For Local Area Networks [HOUS88] describes a solution. According to [HOUS88] every station performs several two-way exchanges with the KDC when powering up. One of the exchanges serves to distribute the broadcast key and the other exchanges serve to distribute the group keys needed by the station.

A	= the station address of the initiator
B	= the broadcast or multicats station address
I	= the nonce
KEK_A	= the key encryption key of A
TEK	= the traffic encryption key
T_{exp}	= the expiration time for TEK
$KEK_A[x]$	= the value encrypted with the key encryption key for A
KID_B	= the traffic encryption key identifier for the TEK
1)	= indicates the order in which steps are performed

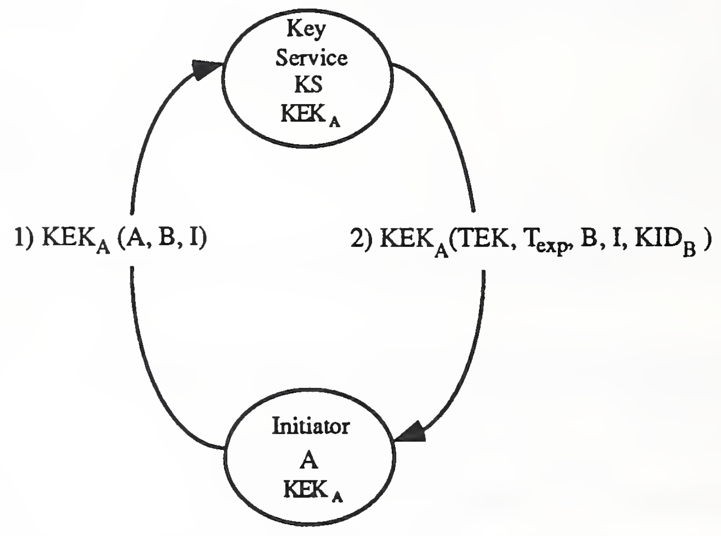


Figure 5.9:1 Distribution of group and broadcast keys

These exchanges will be repeated shortly before the expiration time associated with each traffic encryption key (TEK).

If key management is based on asymmetric principles, there can be a problem creating keys which are shared between more than two parties. For example, Diffie-Hellman is based on the fact that the key is formed by *two* parties.

It should also be mentioned there are mixed feelings about keys shared by several parties. Some security experts have the opinion that a key never should be shared between more than two parties. When more than two parties share a key and a breach in security is detected, it may be difficult to find out who or what caused the breach. If only two parties, Party A and Party B, are involved and Party B causes the breach, Party A could take the necessary precautions. Even if Party B denies to have caused the breach, Party A could be certain that Party B was the guilty part, because Party A knows that it is innocent. However, in the case of SILS, the multiple shared keys are used to protect broadcast and group traffic in a LAN. The need of these keys originates from the broadcast nature of LANs.

5.10 ALGORITHM INDEPENDENCE

ISO security standards should never mention a specific algorithm. One of the requirements for the OSI key management protocol was to keep the protocol independent of specific algorithms. How can this be achieved? An OSI key management protocol residing in the application layer must be specified in Abstract Syntax Notation One (ASN.1) [ISO 8824] [ISO 8825]. We must therefore look at what abilities are provided by ASN.1. For this purpose ASN.1 provides the data type Object Identifier and the parameters associated with an Object Identifier can be referenced using the ASN.1 ANY DEFINED BY construct. Another way to achieve algorithm independence is to alter the presentation context. However, ultimately the algorithm dependent data must be defined somewhere. One solution is to have security registers where the algorithm dependent data is defined. At the moment there are no security registers available. The National Institute of Standards and Technology (NIST) is planning to establish a security register where security objects can be registered. [NIST SEC REG92] provides more information about the NIST security register. [ISO 9979] specifies the procedures for the registering of cryptographic algorithms and the form of register entries. A cryptographic algorithm is an example of a security object that could appear in a security register.

5.10.1 Algorithm independence by using Object Identifiers

An Object Identifier has a certain position in the "Object Identifier tree". The top nodes of the Object Identifier tree are shown below:

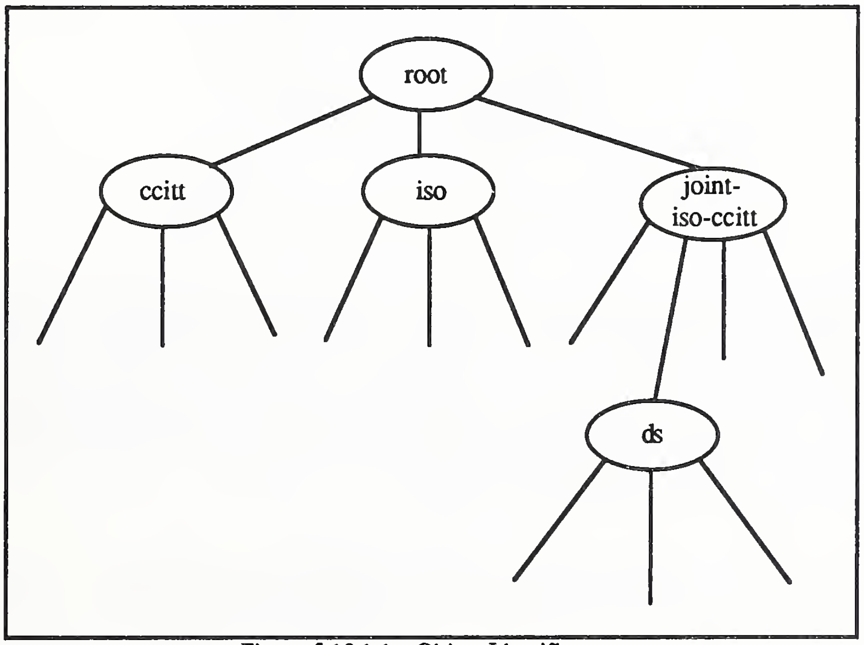


Figure 5.10.1:1 Object Identifier tree

For example, the Directory standards can be referred to as {joint-iso-ccitt ds(5)}, it is the value notation of an ASN.1 OBJECT IDENTIFIER.

References to the security register are possible by using Object Identifier data types. The intention of the following example is to give an idea of how a registration procedure may appear. This example illustrates how DES is registered as an encryption algorithm using the Object Identifier scheme. The algorithm can be registered in the following way:

```
des ALGORITHM
  PARAMETER SEQUENCE {
                        mode INTEGER {ecb(0), cbc(1), cfb(2)},
                        initVector OCTET STRING OPTIONAL}
  := {confAlgs 1}
```

The ALGORITHM ASN.1 macro is defined in the Directory Authentication Framework [ISO 9594-8]. As already mentioned the parameters associated with an algorithm can be referenced using the ASN.1 ANY DEFINED BY construct. The following ASN.1 definitions appear in a protocol that has to be algorithm independent:

```
AlgorithmIdentifier ::= SEQUENCE {
                          algorithm OBJECT IDENTIFIER,
                          parameters ANY DEFINED BY algorithm OPTIONAL}
```

The protocol entity chooses the algorithm by giving the ASN.1 variable *algorithm* the Object Identifier value of the selected algorithm. The field *parameters* contains the parameters associated with the selected algorithm. The ASN.1 data type *AlgorithmIdentifier* is defined in [ISO 9594-8]. The following gives an illustration of what kind of data that is sent when the ASN.1 data type AlgorithmIdentifier appears in an application layer protocol identifying DES used in cbc mode:

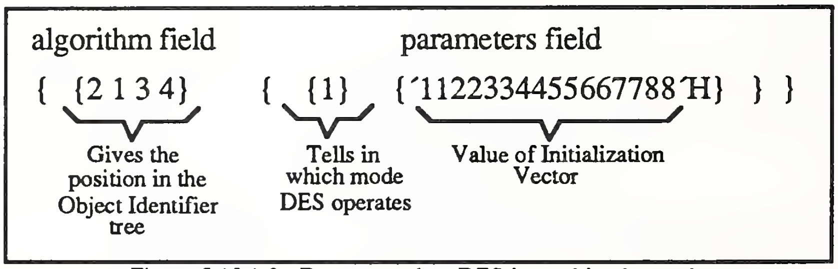

algorithm field
parameters field
{ {2 1 3 4} { {1} {'1122334455667788'H} } }
Gives the position in the Object Identifier tree
Tells in which mode DES operates
Value of Initialization Vector

Figure 5.10.1:2 Data sent when DES is used in cbc mode

By separating from the protocol the parameters relevant for a certain algorithm, nothing algorithm specific needs to be defined in the protocol. This gives a security domain the possibility to choose the algorithms it finds appropriate to use. It also gives the security domain the possibility to use a publicly available key management protocol in combination with its own secret cryptographic algorithms. Another advantage is that the user of the protocol is given the possibility to choose another algorithm when the one being used is broken. As already mentioned, the algorithm specific parameters are defined in a security register.

5.10.2 Algorithm independence by altering presentation context

Another way to achieve algorithm independence is by using the concept of document types. In order to understand how document types can be used, parts of the FTAM standard [ISO 8571-2] [ISO 8571-4] are studied. The similarity between FTAM and key management is that the FTAM protocol has to be independent of a certain file content; the key management protocol has to be independent of a certain algorithm. The FTAM standard solves the problem by having different presentation contexts. A specific presentation context is used when FTAM protocol control information is sent and another when file content data is sent. The presentation context associated with the file content data is determined from a document type. The document type, among other things, describes the content of a file from a syntactic point of view, see [ISO 8571-2] for more information. OSI Explained [HENS88] describes a presentation context. The following description is a more informal way of showing why presentation contexts need to be changed, and gives the reader an idea of how it could be used to support key management. In the following example the ASN.1 tag types, Universal, Application, and Context-specific are used. If a tag is Universal, it can be used in an arbitrary application protocol, the ASN.1 parser always recognizes the associated data type. However, if tags are Application or Context-specific, the parser must know the specific protocol where the tags are used. For example, the following definition from the FTAM protocol [ISO 8571-4] is an example of an Application tag:

```
User-identity ::= [APPLICATION 22] IMPLICIT GraphicString
```

To parse the expression, the parser must know that it has to parse FTAM protocol control information, because in another application protocol [APPLICATION 22] may be used to tag something that is completely different from the above definition.

Context-specific tags are also used in [ISO 8571-4]. For example:

```
F-RECOVER-request        ::= SEQUENCE {
   activity-identifier      Activity-Identifier,
   bulk-transfer-number     [0] IMPLICIT INTEGER,
   requested-access         Access-Request,
   access-passwords         Access-Passwords OPTIONAL,
   recovery-point           [2] IMPLICIT INTEGER DEFAULT 0,
   remove-contexts          [3] IMPLICIT SET OF Abstract-Syntax...,
   define-contexts          [4] IMPLICIT SET OF Abstract-Syntax...}

F-RECOVER-response       ::= SEQUENCE {
   state-result             State-Result DEFAULT success,
   action-result            Action-Result DEFAULT success,
   contents-type            [1] Contents-Type-Attribute,
   recovery-point           [2] IMPLICIT INTEGER DEFAULT 0,
   diagnostic               Diagnostic OPTIONAL,
   presentation-action      [6] IMPLICIT BOOLEAN DEFAULT FALSE}
```

Now the parser must not only know that it has to parse FTAM protocol control information; it must also know which FTAM PDU it has to parse. Otherwise, it cannot distinguish between Context-specific tags with the same value. In the above definitions both PDUs use [2] as a Context-specific tag. When the presentation context is altered, the parser knows that is has to parse another type of ASN.1 information that it has specified in a document type, where eventual Application and Context-specific tags are associated with other data types than the ones when FTAM protocol control information is sent. When FTAM´s initiator and responder exchange the initialization PDUs, they can also agree about what document types to support. For key management, the algorithm specific data could be described in a document type; the document type could be part of a security register. When the presentation context is changed, the parser knows from having parsed key management protocol information. It has to parse information associated with a document type. The document type, among other things, describes the ASN.1 data types associated with a particular key exchange algorithm.

5.10.3 Algorithm independence in SDNS

[SDNS 90-4262] uses the ASN.1 OCTET STRING data type to keep the protocol independent of specific cryptographic algorithms. Another benefit is that the semantics of the data is not revealed. An example is the Nkrq PDU, which has the following ASN.1 definition:

```
Nkrq ::= SEQUENCE {
            init-cred Credentials,
            univ-id Universal-id,
            init-kid KeyId}
Credentials ::= [APPLICATION 22] IMPLICIT OCTET STRING
```

It is obvious that the real ASN.1 definition of credentials must be something else than OCTET STRING. How the ASN.1 parsing is done is not clear, one way could be to do the parsing in two steps. The first step handles the data like an ASN.1 OCTET STRING data type and the ASN.1 OCTET STRING identifier tag and length information are removed. The second step does the parsing in accordance to the real representation of the data. This representation will also make it possible to use the same protocol in different environments where the representation of Credentials may vary.

5.10.4 Section summary

The advantage of the method where the presentation context is altered is the possibility to define a state machine for each particular algorithm. It makes it possible for the algorithm designer to specify the handshaking procedure. For the Object Identifier approach, if the scheme of SDNS is followed, the algorithm dependent information is carried by the Nkrq, Nkrs, Ssrq and Ssrs PDUs, i.e. the handshaking can only be four-way.

If it is not possible to mix protocol control information and algorithm dependent information, because they have to be parsed in different presentation contexts, a drawback is that the number of transitions increases. For example, if the key exchange algorithm described in NLSP [ISO 11577] is used in an application layer based key management protocol (see Annex A for more details about the algorithm) and algorithm independence is achieved by changing presentation context, algorithm dependent data cannot be carried by the Ssrq and Ssrs PDUs. This is because the Ssrq and Ssrs PDUs can only carry the key management protocol control information. Therefore, at least three transitions have to occur in the presentation context where algorithm dependent data is exchanged.

The solution employed in SDNS can be seen as a solution designated for that particular environment and is not applicable for the general case.

5.11 ASN.1 ABSTRACT VERSUS TRANSFER SYNTAX PROBLEMS

When security has to be employed in the application layer, ASN.1 causes some problems. The problems listed in this section are not specific for a layer seven key management protocol, they are general ASN.1 problems encountered when security services reside in the application layer.

The content of an an application layer protocol is specified by using ASN.1 [ISO 8824] [ISO 8825]. The way in which the ASN.1 definitions are represented internally within a system is a local matter. When two systems exchange PDUs they must, of course, have the same external view about the representation of the PDU content. According to the OSI model, it is the presentation layer that translates from a local representation to the common ASN.1 format. The common format is the ASN.1 transfer syntax representation of the data and the local format is the abstract syntax representation. In a real implementation the place where the translation is done may vary. For example, in a real implementation the presentation and application layers are often strongly coupled. The choice of how to represent the abstract syntax affects both layers.

Digital signatures must be computed on a data structure which both parties have in common. Otherwise, it would not be possible for the receiver to verify the signature. Thus, since the signature cannot be computed on the abstract syntax representation it must instead be computed on the transfer syntax representation. The problem is that the transfer syntax of the data may not be available when the signing occurs, since the translation from abstract to transfer syntax can be done when the data is processed in the presentation layer. The problem can be solved by doing a translation immediately before the signing occurs. For example, ISODE [ROSE90] represents the ASN.1 elements of the application layer as *presentation elements*. The presentation layer translates the presentation elements representation to an octet stream, which is the ASN.1 transfer syntax representation. When signing needs to occur in the application layer, the translation to the transfer syntax representation could be done and the hashing and encryption performed on the transfer syntax representation of the data. However, after the data is sent to the presentation layer the same translation will be done once more. In order to avoid retranslations the existing ASN.1 processing schemes must be altered. If an existing ASN.1 parser, in conjunction with a presentation layer implementation is used, probably problems are encountered when implementing security services in the application layer.

Distinguished encoding is another problem. For example, the ASN.1 length information can be represented in different ways. The Directory standards makes certain restrictions about how the ASN.1 transfer syntax can be encoded when there are multiple choices for representing ASN.1 transfer syntax. In some way the ASN.1 parser must be forced to choose a specific transfer syntax representation. For example, routing in the application layer may cause problems. Suppose that a signed ASN.1 data unit is sent from OSI system A to C but passes through the application layer of system B. The receiving system, C, when checking the digital signature may compute a hash value that differs from the one provided. This occurs because after being sent from the application layer, the signed data unit is encoded in the presentation layer by system B using a different scheme. The length representation of the data unit sent from system A was chosen by system A´s ASN.1 parser, however, the length representation of the data unit sent from system B was chosen by system B´s ASN.1 parser. If distinguished encoding is used this problem will not occur.

When most ASN.1 parsers were designed, the special problems encountered when employing security in the application layer were not considered.

5.12 TRIGGERING OF KEY MANAGER

A key manager residing in the application layer could be triggered from the ACSE when an application association is established. A drawback to this approach is that the key manager needs to be retriggered when a key update (key update is defined in section 5.19) has to be done. The retriggering cannot be done from the ACSE, because the ACSE is only involved when an application association is created or terminated. Another solution is to trigger the key manager from the security protocol, as shown below:

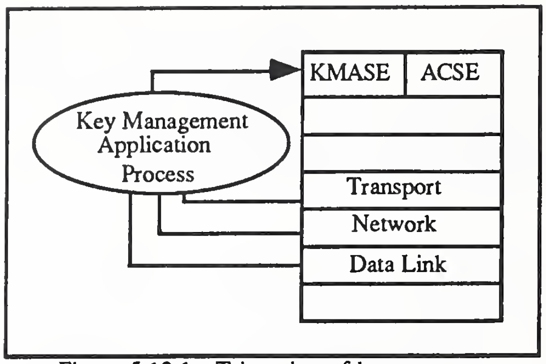


Figure 5.12:1 Triggering of key manager

It should be pointed out that earlier this type of interaction was modelled explicitly as part of systems management. It was the responsibility of the layer manager entity (LME) to accomplish such things as triggering. However, in OSI the concept of layer management entities does not exist any more. The interaction between the security protocol and the layer manager was described as shown in the following figure:

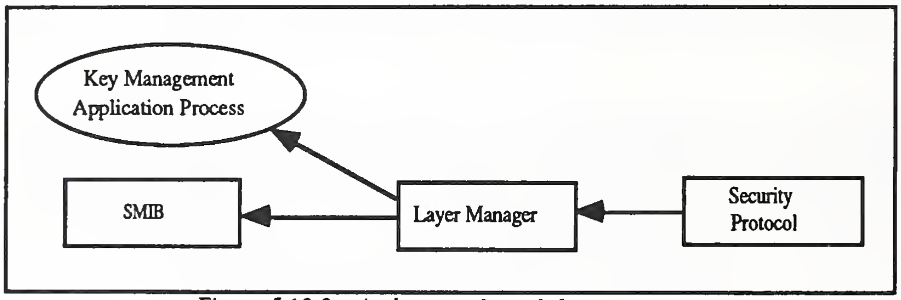


Figure 5.12:2 Actions go through layer manager

The security protocol entity, according to this model, invokes the layer manager entity in order to get the cryptographic keys. The layer manager entity in turn checks the SMIB to determine whether there are any keys available. If there are no keys available the layer manager triggers the key manager.

Triggering can be done in the following way when TLSP (former SP4C) is used on a per connection basis:

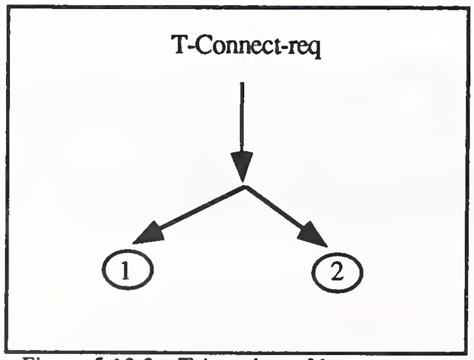


Figure 5.12:3 Triggering of key manager

When a T-Connect request service primitive arrives and security services are to be employed, the following occurs:

1. The key manager is triggered and the needed keys and attributes are established. This can be done over an unprotected transport connection, because the key manager relies on security services in the application layer.
2. Transport sends a protected CR PDU.

TLSP security services are offered either on a per transport connection or on a per NSAP basis. When the security services are offered on a per NSAP basis it is possible for several transport connections to share the same cryptographic keys. In this case when a new secure transport connection has to be established, a check could be performed to find out whether an appropriate security association is already established with the peer entity. If so, there is no need to trigger the key manager.

Problems may occur when the key manager is triggered from the security protocol. First, how can it be determined that the particular transport connection transferring key exchange data shall not be protected, when the local security policy mandates that security services are invoked when communicating with the other system? Second, if the key management entity is associated with a particular transport selector, how is it determined that it really is the key management entity that sends the data? The data could be sent by another entity trying to bypass the security mechanisms of the system?

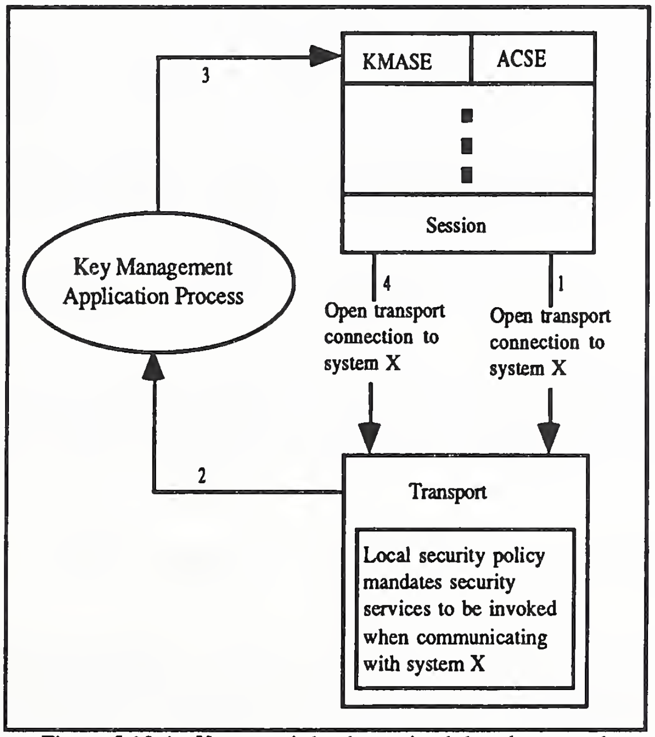

KMASE
ACSE
Session
Key Management Application Process
3
4
1
2
Open transport connection to system X
Open transport connection to system X
Transport
Local security policy mandates security services to be invoked when communicating with system X

Figure 5.12:4 How can it be determined that the second transport connection does not need to be protected

The figure above illustrates the steps involved in triggering the key manager. The second transport connection (see step 4) does not need to be protected because of two reasons: First, it is not possible to protect the transport connection since the transport connection is used to establish the traffic encryption keys. Second, the data is protected by security mechanisms employed in the application layer. The problems mentioned may not occur in SDNS because there is a separation between the different stacks. [LAMB88b] provides description of the dual and triple stack approaches. The following figure illustrates a triple stack:

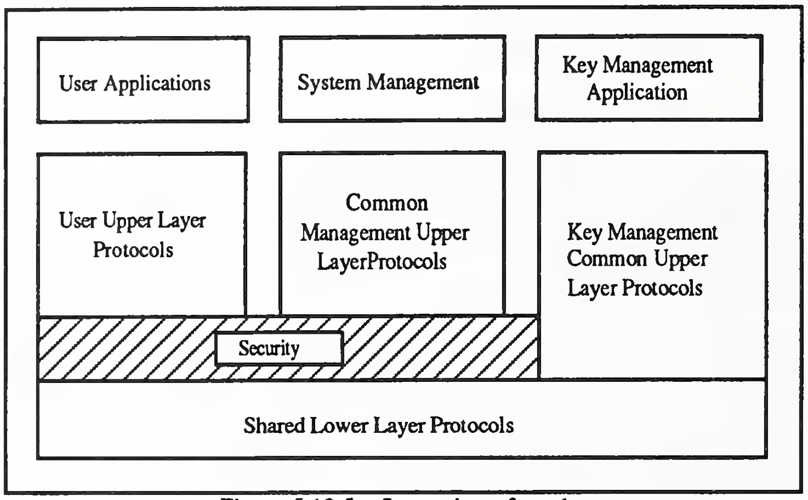

User Applications
System Management
Key Management Application
User Upper Layer Protocols
Common Management Upper LayerProtocols
Key Management Common Upper Layer Protocols
Security
Shared Lower Layer Protocols

Figure 5.12:5 Separation of stacks

Security services provided by TLSP reside within the user and management stack. The transport layer of the key management stack is separate from transport layer of the user and management stack. Automatically it is known that a transport connection of the key management stack does not need to be protected. Since there is a separation of stacks, only the key management entity is able to send the data. However, if in a real implementation there is no separation of stacks; the problem of how to assure that an entity is not bypassing the security mechanisms of the system remains unresolved. One solution to this problem is requiring a trusted implementation of the protocol at an appropriate level in the Trusted Computer System Evaluation Criteria [DoD 5200.28].

Another problem with triggering the key manager from the security protocol is that existing network service interfaces, for example, [x/OPEN TI], do not provide any key management triggering parameters.

If the triggering is done from the security protocol it is easier to accomplish super encryption (for example, encryption could be employed at both an upper and lower layer), because the security protocols involved can interact with the key manager independently of each other.

5.13 ESTABLISHMENT OF SECURITY SERVICES

How can the key manager be informed about the security services and associated security levels that a security protocol entity wants to establish? This is related to the problem of how an application or user can inform the security protocol entity which security services to invoke. Often two different solutions to the problem are mentioned: by local systems management and Quality of Service (QOS) parameters.

An example of local systems management may be a configuration table specifying the security services to be used when communicating with other systems, the security levels allowed, during which times it is allow to communicate, etc. It is configured in accordance with the local security policy.

The Upper Layer Security Model document [ISO/SC21 6095] mentions that the only way by which an application can influence the selection of Lower Layer security services for a particular instance of communication is by using security QOS.

The Lower Layer Guidelines document [ISO/SC6 6957] introduces the *protection* Quality of Service parameter (SQOS). Note, some security documents use the term *security* QOS while other security documents use the term protection QOS. The term security QOS seems to be used when the discussions concern security relevant QOS parameters in general. For example, both the *TC protection* QOS parameter as defined in the Transport Service definition [ISO 8072] and the protection QOS parameter as defined in [ISO/SC6 6957] can be included. While the term protection QOS is used in discussions concerning the protection QOS parameter as defined in [ISO/SC6 6957]. The views presented in [ISO/SC6 6957] are that the security part of the ordinary QOS parameter as, for example, the TC protection QOS parameter, is insufficient for security purposes. The SQOS parameter is passed between layers and one way of determining the security services to be included in the security association is:

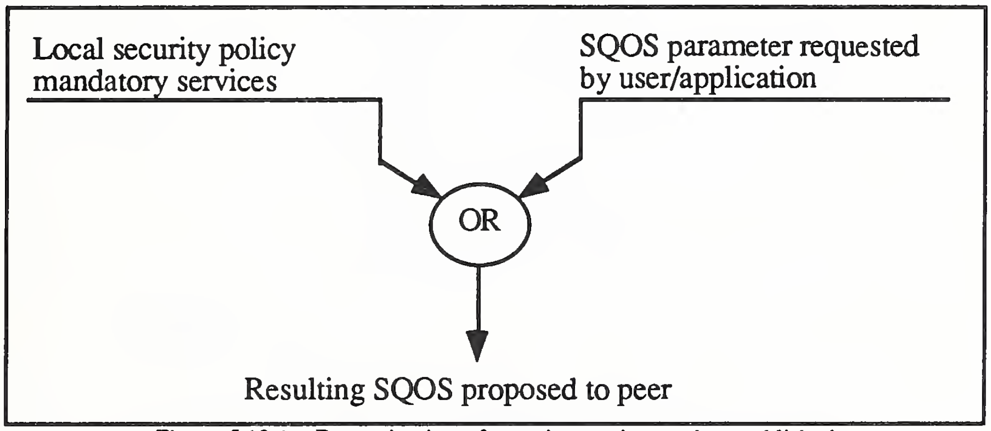

Local security policy mandatory services
SQOS parameter requested by user/application
OR
Resulting SQOS proposed to peer

Figure 5.13:1 Determination of security services to be established

For example, the *local security policy* may stipulate that when communicating with a specific system the communication must be integrity protected. In a commercial environment where only a small portion of the communication needs to be protected, it could be the responsibility of the *application or user* to invoke the security services on an as needed basis.

NLSP-CO specifies the following protection QOS sub-parameters:

- Peer Entity Authentication;
- Access Control;
- Connection Confidentiality;
- Traffic Flow Confidentiality;
- Connection Integrity without Recovery;
- Security Label.

The SQOS parameter can be mapped onto the Service Selection field (except the Security Label) of the NLSP Security Control Information (SCI) PDU. If the Security Label is used it is possible to map the security level expressed by the label to the parameters included in the Service Selection field. A specific protection QOS sub-parameter, e.g. Connection Confidentiality, may be expressed as an integer. The value 0 may imply that no confidentiality service needs to be invoked and value 1 that confidentiality algorithm X has to be used. The information which needs to be preestablished is included in what the NLSP document [ISO 11577] calls An Agreed Set Of Security Rules (ASSR). So the confidentiality algorithm(s) associated with the SQOS sub-parameter Connection Confidentiality must be defined in the ASSR.

By sending the SQOS parameter between different layers, the user informs the NLSP entity of the security services to be established. By sending the SCI PDU the peer NLSP entity is informed about the security services to be included in the security association. However, [ISO 11577] does not explicitly state whether it is possible to enhance security at the NLSP sublayer or whether it is possible for the responder to enhance security at connection establishment. [ISO 11577] states that it is assumed that the NLSP entity attempts to provide security as defined in a target protection SQOS. The following is therefore just an interpretation of how the SQOS parameter may be used.

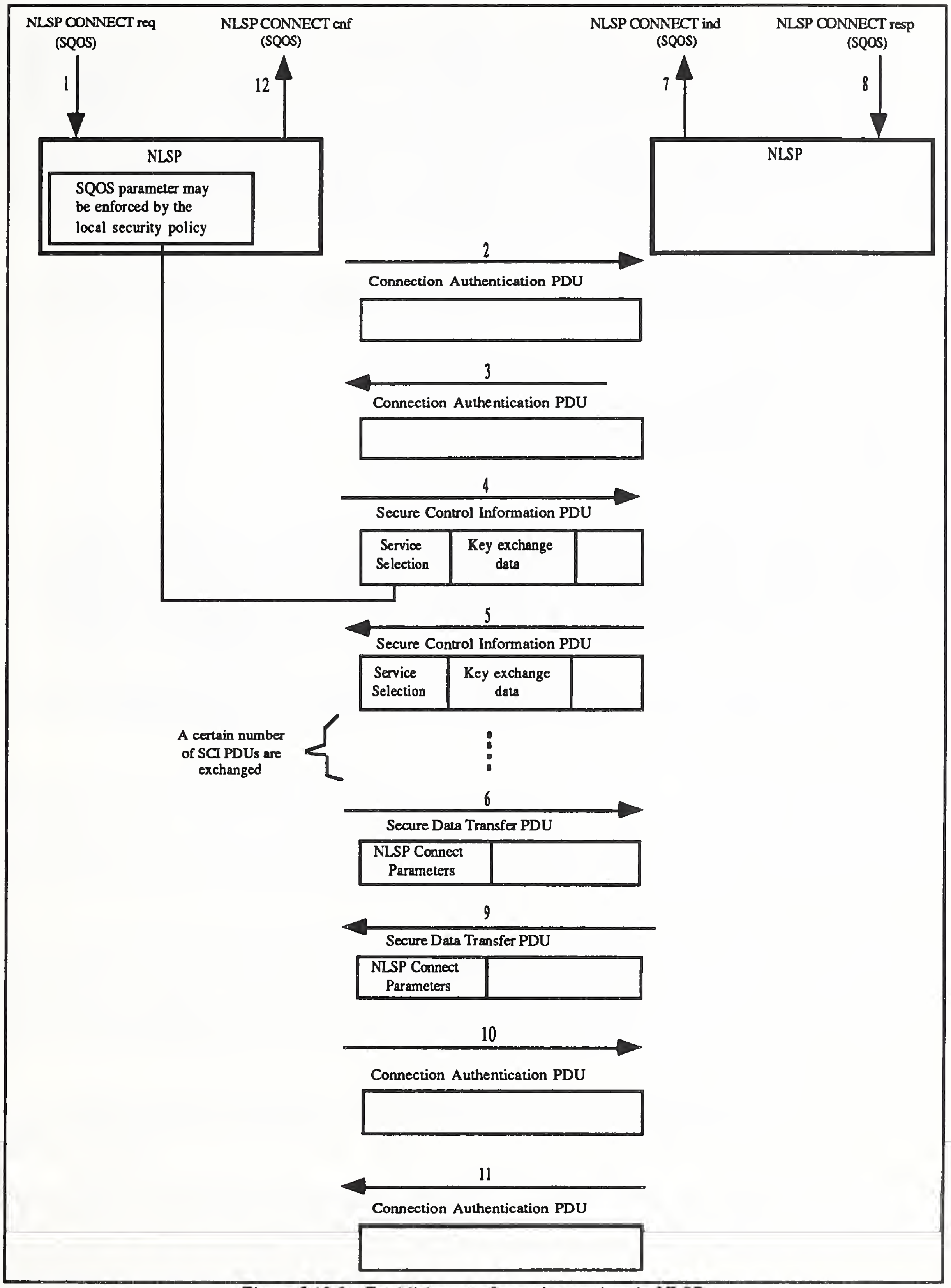


Figure 5.13:2 Establishment of security services in NLSP

If the service provider, NLSP, cannot meet the SQOS requirements an NLSP Disconnect service primitive is sent to the service user. In Figure 5.13:2, the cryptographic keys and attributes needed for the security association are established before an NLSP Connect indication is issued at the responding side. The service user at the responding side may want a higher level of security; and if so, the responding user sends an NLSP Disconnect request service primitive *after* the keys and attributes have been established. The SQOS parameter in this example can be enhanced by security administration imposed security policies. The Secure Data Transfer PDU sent by the initiating entity after the last SCI PDU contains the SQOS parameter specified by the user of the NLSP entity at the initiating side.

The QOS parameter defined in [ISO 8072] which can be used for security purposes is *TC protection*. The parameter is specified by selecting one of four protection options, these options are:

1. no protection features;
2. protection against passive monitoring;
3. protection against modification, replay, addition or deletion;
4. both 2 and 3.

Choosing 1 implies no security services at all; alternative 2 implies a connection oriented or connectionless confidentiality service; alternative 3 implies an connection oriented or connectionless integrity service; alternative 4 implies both a connection oriented or connectionless confidentiality service and a connection oriented or connectionless integrity service. The value of *TC protection* must in the T-CONNECT indication be the same as in the T-CONNECT request.

The following is an example of how the TC protection parameter may be used to support a layer seven key manager. This example assumes that the security services are established on user demand.

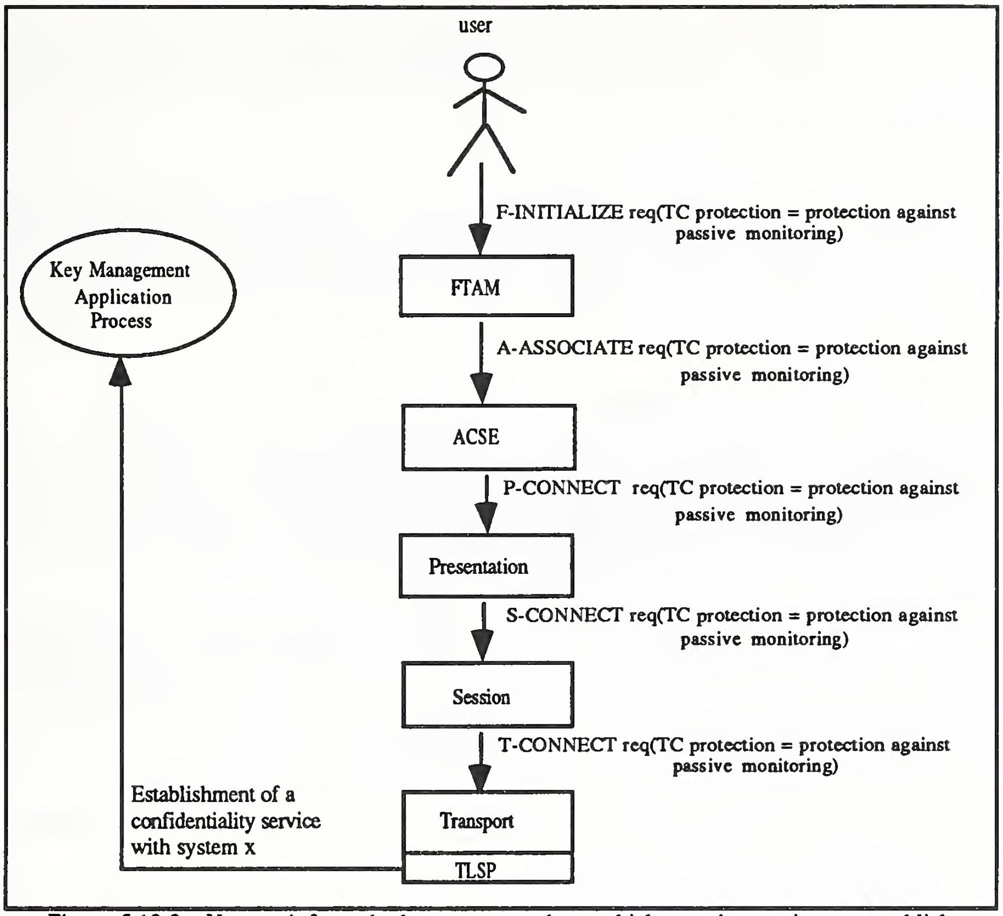

user
F-INITIALIZE req(TC protection = protection against passive monitoring)
FTAM
A-ASSOCIATE req(TC protection = protection against passive monitoring)
ACSE
P-CONNECT req(TC protection = protection against passive monitoring)
Presentation
S-CONNECT req(TC protection = protection against passive monitoring)
Session
T-CONNECT req(TC protection = protection against passive monitoring)
Transport
TLSP
Key Management Application Process
Establishment of a confidentiality service with system x

Figure 5.13:3 How to inform the key manager about which security services to establish

Should it be allowed for the responding key manager to enhance the security? For example, if the local key manager wants to establish a confidentiality service, the peer key manager may also want to include an integrity service. It is not possible for the user in Figure 5.13:3 to inform the security protocol about the security level. For this purpose a security label, which is included in the SQOS parameters defined in [ISO 11577], could be used.

How is a user guaranteed that the SQOS parameter is not bypassed? The user may believe that security services have been invoked which have not been invoked, because the SQOS parameter is bypassed somewhere in the path between the application protocol and the security protocol. One solution to the problem is again to use a trusted implementation of the protocol at an appropriate level in the Trusted Computer System Evaluation Criteria [DoD 5200.28].

5.14 KEY MANAGEMENT AS PART OF THE SECURITY PROTOCOL

One problem with having key management as part of a security protocol is that the communication service offered to the security protocol may be connectionless. In some of the following discussions it is assumed that TLSP [ISO 10736] resides in a class four transport protocol and the underlying network is connectionless.

By definition a connectionless service provides no guarantee that a particular PDU is delivered. For example, it is obvious that a Diffie-Hellman exchange cannot occur easily. When the traffic encryption keys are formed, by using the Diffie-Hellman algorithm, Party A has to send the following to Party B:

$\alpha^X \bmod \rho$.

However, there is no assurance that the data reaches Party B, because there is no guarantee of delivery. Party A could continue to resend the data until an acknowledgement from Party B is received. Thereafter Party B has to send the following to Party A:

$\alpha^Y \bmod \rho$.

The procedure will now be repeated with the roles of Party A and Party B changed.

The NLSP document [ISO 11577] states that if the expected sequence of PDUs does not occur within a specified timeout then SCI exchange PDUs may be repeated any number of times. The problem of having a connectionless network can therefore be solved. *However, if there is a class four transport residing in the system, its functionality is duplicated.*

There have been discussions about including a key exchange capability in TLSP. Two different solutions have been discussed. The first solution is based on having a key management entity acting as a transport user. As we will see later this solution has several disadvantages and was therefore dismissed. To avoid the problems with a connectionless network a key management entity is acting as user of the transport service, see the following figure.

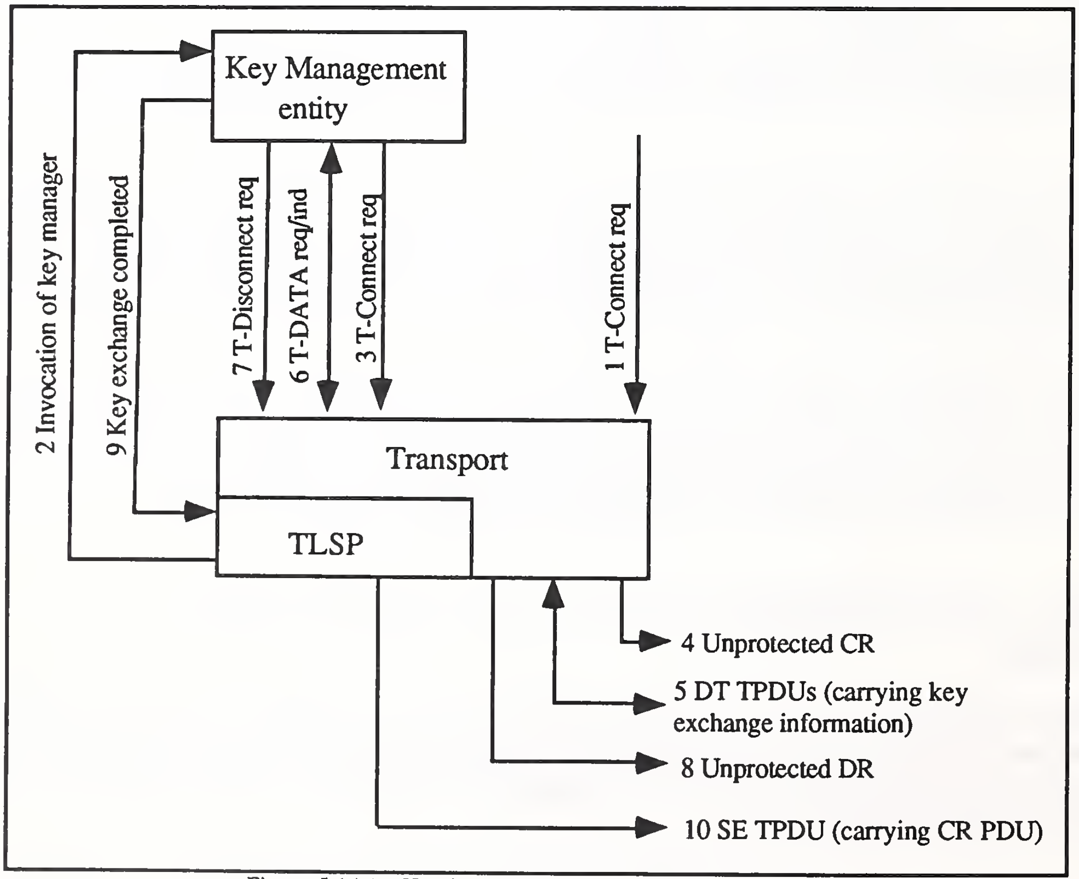


Figure 5.14:1 Key Management Entity as Transport user

The disadvantages of this solution are:

1. To have a key management entity acting as transport user is not consistent with the OSI model since the transport user has to be the session layer entity.
2. A lower layer (TLSP) uses the services of a higher layer (key management entity), in order to obtain reliable end-to-end data transfer. The ordinary case is that a higher layer uses the services provided by its lower layer.
3. The solution works only for the connection oriented transport protocol not the connectionless.

The second solution is based on including a key exchange capability in TLSP [ISO 10736]. Because of the difference between TLSP and NLSP it is more difficult to extend [ISO 10736] with a key exchange capability. TLSP is an extension of the transport layer, while NLSP is an "independent" sub-layer at the top of the network layer. An NLSP entity is invoked by service primitives, see Figure 5.14:2. It may look appealing to consider having TLSP at the top of the transport layer instead, so that it could rely on a connection oriented service[13]. However, the sequence numbers of the transport entity would not be protected then, therefore connection integrity would be harder to achieve.

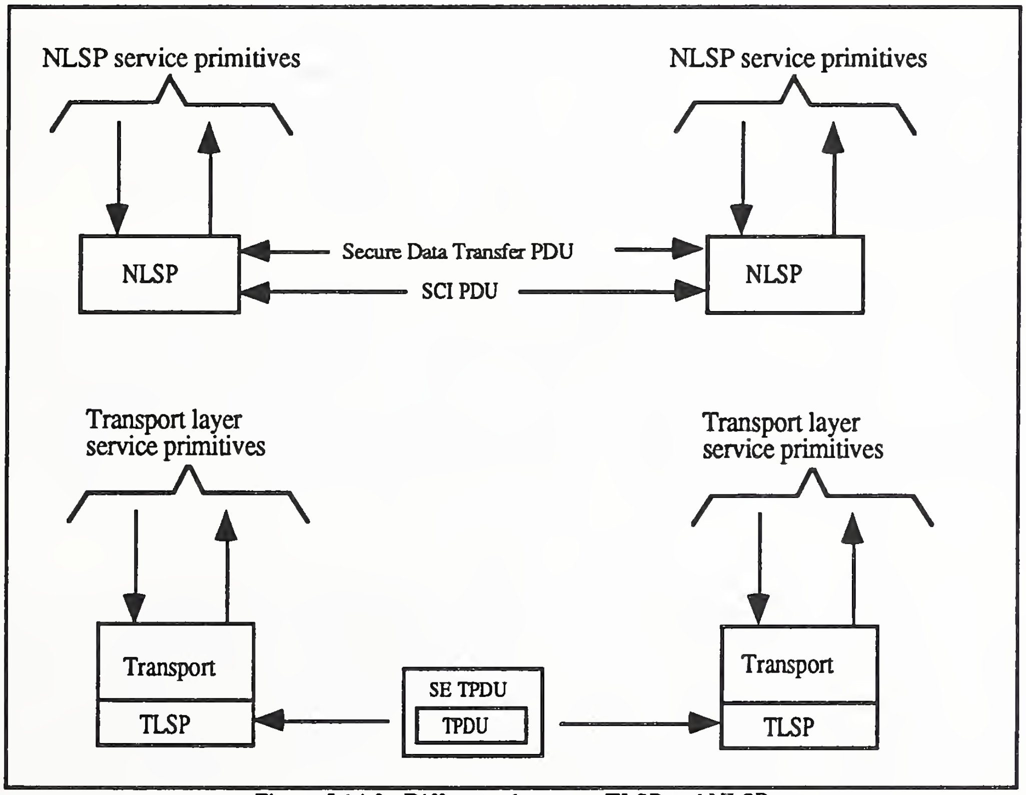


Figure 5.14:2 Difference between TLSP and NLSP

[13] Provided it is the connection oriented transport protocol.

The SQOS parameter included in the NLSP Connect request (possibly enhanced) is mapped onto the SCI PDU. The user data is sent after the cryptographic keys and attributes are established. The user data is therefore not sent in the clear. If the SQOS parameter, needed to inform peer of requested SQOS, is mapped onto the CR PDU of the transport protocol, the CR PDU must be sent in the clear. This is because the CR PDU is used to inform the peer entity about the security services to include in the security association. It may not be acceptable to send the CR PDU in the clear. See also the next section 5.15. In order to avoid to send the CR PDU in the clear, TLSP needs special PDUs. The following illustrates the problems involved if TLSP were enhanced with the PDUs needed to establish a security association and the associated keys and attributes.

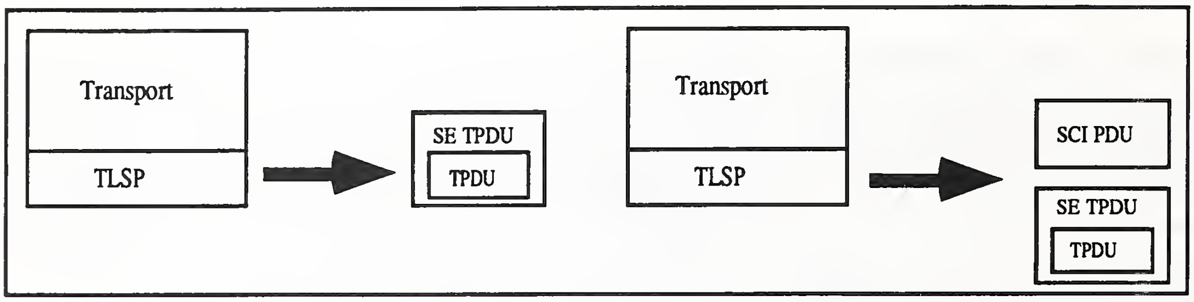


Figure 5.14:3 Two different ways to include TLSP in the transport layer

On the left side of Figure 5.14:3, TLSP is integrated with the transport protocol; therefore the TLSP entity is just encapsulating TPDUs. If the TLSP entity encapsulates a TPDU and the encapsulated TPDU never reaches the receiver, the sending transport entity retransmits the TPDU and the TLSP entity encapsulates the TPDU again. TLSP does not need to be aware of any retransmission mechanisms since it is integrated in the transport protocol. If a security encapsulated TPDU gets out of order, the receiving TLSP entity does not need to take any measures. The measures are taken by the transport entity when the TPDU is processed.

On the right side of Figure 5.14:3, the TLSP entity is issuing SCI PDUs which do not encapsulate a TPDU. This scheme makes TLSP more loosely coupled with transport and therefore it is not possible for TLSP to rely on the mechanisms of the transport protocol. If the network offers a connectionless service, there is no guarantee of delivery. Because the underlying network has no reliable delivery, TLSP must include retransmission and acknowledge mechanisms. This is a duplication of the transport functionality, a reinvention of transport at the bottom of itself. Despite all the disadvantages listed in this paragraph, this solution is the one adopted by ISO, see [ISO 10736/P].

When performing what [ISO 11577] calls asymmetric and exponential key exchanges, certificates are needed. No explanations are given in [ISO 11577] about the entities to be authenticated by using the certificates, holders of the certificates, etc. Are certificates as described in the Directory standards appropriate for use in the network layer? The certificates are described in ASN.1 which is supposed to be unknown below the presentation layer. If Directory certificates are allowed, they probably would be represented in the ASN.1 transfer syntax form. Examples given in the Authentication Framework [ISO 10181-2] of entities to be authenticated in the network layer are: subnetworks, network nodes and relays. Therefore it is assumed that those are the entities to be authenticated when an NLSP key exchange occurs.

5.15 KEY MANAGEMENT AS PART OF A LAYER PROTOCOL

A possibility is to convey key management information in the Connect Request (CR) and Connect Confirm (CC) TPDUs of the Transport protocol, see A Key Management Algorithm For Secure Communication In Open Systems Interconnection Architecture [RAMA90]. The protection parameters included in the variable part field of the CR and CC TPDUs headers would contain the information.

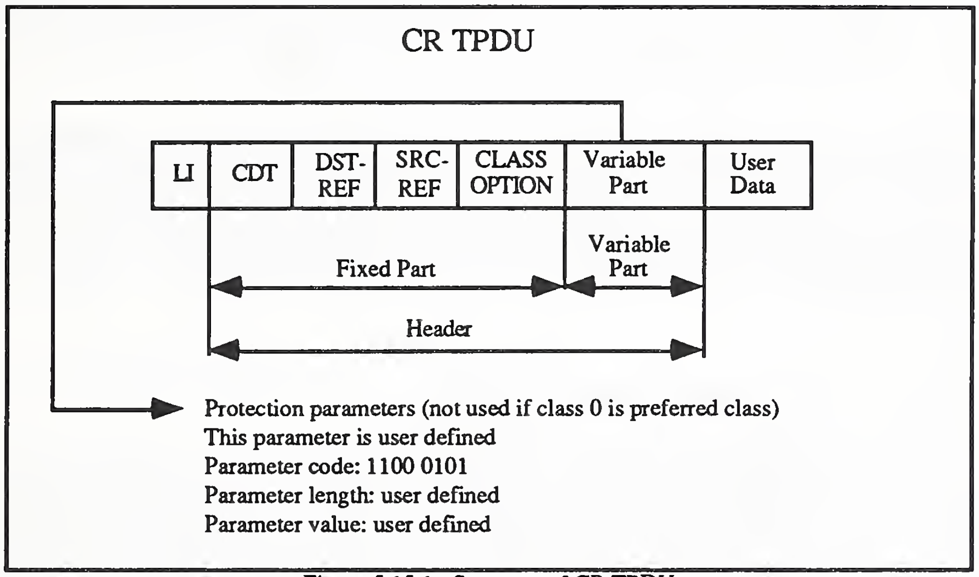

CR TPDU
LI
CDT
DST-REF
SRC-REF
CLASS OPTION
Variable Part
User Data
Fixed Part
Variable Part
Header
Protection parameters (not used if class 0 is preferred class)
This parameter is user defined
Parameter code: 1100 0101
Parameter length: user defined
Parameter value: user defined

Figure 5.15:1 Structure of CR TPDU

If this solution is adopted the key exchange is done when the transport connection is established. A big problem is the limitation in size of the information that can be included in the variable part field. [RAMA90] mentions that the CR TPDU header can consist of 254 octets. However, the transport protocol definition [ISO 8073] specifies that the length of the CR TPDU shall not exceed 128 octets, despite the fact that the maximal possible value of the length indicator is 254. From the available 128 octets, those needed for the transport protocol control information have to be subtracted. The remaining octets can be used to carry key exchange information. Certificates, and especially certification paths, as described in the Directory Authentication Framework contain much more information than can be conveyed in the variable parts field of CR and CC TPDUs. The handshaking procedure explained in [RAMA90] can only be two-way, which may be a severe limitation. For example, the two way exchanges explained in the the Directory Authentication Framework [ISO 9594-8] and [RAMA90] have in common that they use timestamps. Therefore, the algorithm designer must choose a specific solution and maybe not the solution he or she prefers, because the preferred solution cannot be realized by a two-way handshake procedure. The CR and CC TPDUs must be sent in the clear; this may not be acceptable. For example, ISODE [ROSE90] uses the TSAP ID parameter, included in the variable part of the header, to determine the application. The TSAP ID parameter could provide useful information to the one passively monitoring the traffic and other approaches offer protection for this information.

5.16 ADDING A NEW COMPONENT TO THE NETWORK

When a new component is added to the network how can it obtain its key material? By key material we mean:

- in an asymmetric form of key management the private key and public key, the latter is usually bound with an identity in the form of a certificate;
- in a symmetric form of key management the Key Encrypting Key.

The following paragraphs only discuss asymmetric considerations. Symmetric key management has been used for many years so organizations have well established procedures to solve the problem.

This issue is often not addressed in standardization work other than by a paragraph saying that credentials are established out of band. It can be regarded to be part of an OSI key management infrastructure but not of an OSI key management protocol. As in the case of rekeying; there may be an advantage if it can be automatized. Little in this area can be found in the literature. One often cited article is Initialization Of Cryptographic Variables In A Network With A Large Number Of Terminals [MATY86].

5.16.1 Adding a new component in SDNS

In SDNS, a seed key is used for this purpose. With a seed key, an SDNS component is allowed to communicate with the KMC but not with another SDNS component. The article SDNS Services And Architecture [NELS87] mentions that the seed key initially is physically distributed from the KMC. When the newly added component contacts the KMC, the seed key is converted to an operational key. The article does not describe in detail what other kinds of information are delivered from the KMC to the new component.

5.16.2 Public key registration

The article Initialization of Cryptographic Variables In A Network With A Large Number Of Terminals [MATY86] describes how the public key of a terminal can be registered at a KDC. The method described is general and can be used to register the public key of entities other than terminals. The method can be used to register the public key of a new component added to a network. The public key is registered by using the network to transmit the needed data. In the scenario described in [MATY86], a terminal has a cryptographic facility (CF) where the private and public keys are generated. The private key never leaves the CF only the public key is displayed, so it can be distributed by security personnel. A prerequisite is that the public key of the KDC and the terminal identification number (TID) are available in the CF. The entity responsible to carry out the steps needed to register the public key is called the terminal initializer. The security of the procedure rests on the assumption that the terminal initializer complies with the issued instructions and understands that failure to comply with these instructions may result in an adversary successfully registering a key with the KDC. [MATY86] mentions that the terminal initializer has no responsibility for transporting keys, public or private, or for installing private keys by entering them directly into a cryptographic device. Therefore, the terminal initializer is not a courier and does not perform the functions of a courier. The one who might perform the duties of a terminal initializer are: a terminal user, terminal owner, manager or member of the local site security. The terminal initializer is provided with a set of instructions outlining the terminal initialization procedure. The terminal initializer is also provided with two expiration times T_1 and T_2, respectively. Prior to the first time T_1, the public key of the terminal must be temporarily registered at the KDC under the designated TID. After the expiration of the second time, T_2, the public key is considered to be permanently registered. If an invalidation occurs after T_2, the identity must be proven. After T_2 and upon request a certificate can be issued for the TID, whose public key has been succesfully registered.

The registration procedure can be divided into six separate steps. It is assumed that the initialization instructions are received at time T_0. The following description of the different steps refers to the state of the KDC, the state diagrams of both the KDC and the terminal, whose public key is registered, are described in [MATY86].

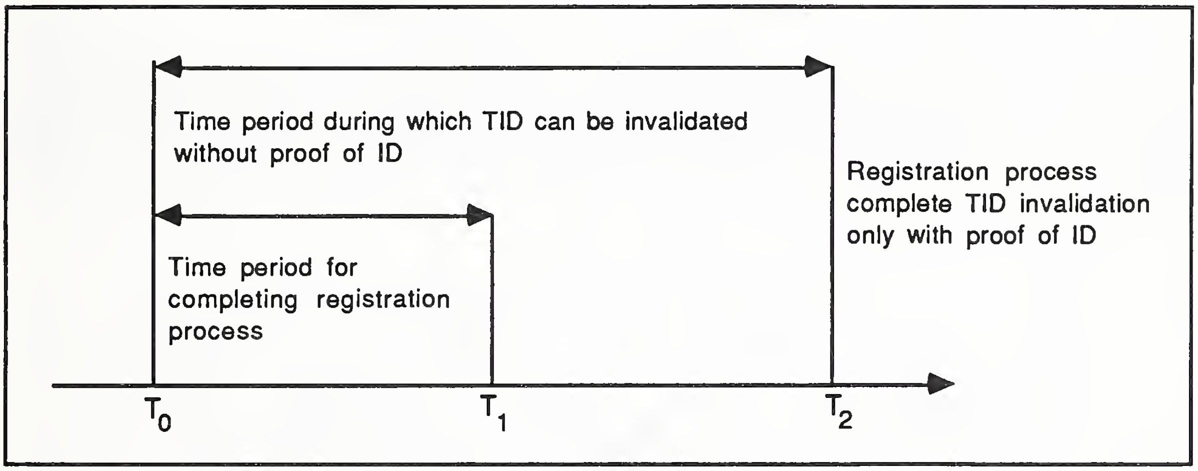

Time period during which TID can be invalidated without proof of ID
Registration process complete TID invalidation only with proof of ID
Time period for completing registration process
T_0
T_1
T_2

Figure 5.16:1 Times in public key registration procedure

Step 1: This step concerns the notification letters which the terminal initializer receives from the KDC.

Step 2: The terminal initializer causes the generation of the public and private keys by using CF.

Step 3: A public key registration message containing the TID, public key, and time variant data is sent to the KDC. The KDC sends a response message containing the state of the KDC, the outcome of processing the public key registration message, the public key registered or temporarily registered for the requested TID, the requested TID and some time variant data. A digital signature is calculated on the response message. By checking the digital signature the requestor can be convinced that the message originates from the KDC and has been received without modification.

Step 4: The terminal initializer performs a verification that the terminal´s public key has been registered at the KDC following the second expiration date. An ID verification message is sent to the KDC containing the TID and time variant data. The response from the KDC contains the TID, optionally expiration times T_1 and T_2, the public key registered or temporarily registered, and some time variant data. A digital signature is computed on the response message. Alternatively, the KDC could return a certificate; the certificate would also serve as proof to the terminal that the public key has been registered.

Step 5: The terminal can invalidate the TID without proof of identity. This step can only occur when the current date and time is prior to the second expiration date T_2. The

ID validation message contains the TID and some time variant data. The response message contains the state of the KDC, the outcome of processing the ID invalidation message, and the same time variant data as was included in the request message. A digital signature is computed on the response message.

Step 6: This step is used for ID invalidation with proof of identity to minimize potential disruption and denial of service. This step must be performed whenever the current date and time is past the second expiration time, T_2.

The article also describes the different error situations that can occur.

5.17 COMMON KEY MANAGEMENT PROTOCOL

Can the same protocol provide both an asymmetric and symmetric form of key management, or must two separate protocols coexist? It is, of course, an advantage if the same protocol provides both forms of key management. One way to achieve this is to separate the first phase, i.e., the key establishment phase according to the class of cryptographic system desired. Later phases could be in common.

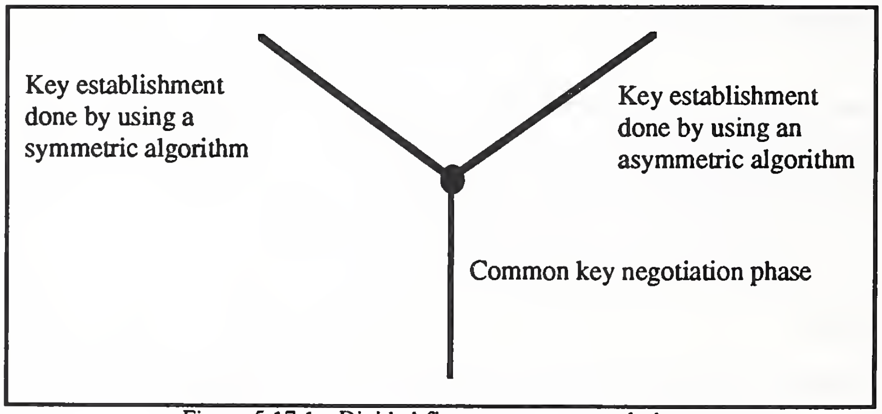

Key establishment done by using a symmetric algorithm
Key establishment done by using an asymmetric algorithm
Common key negotiation phase

Figure 5.17:1 Divided first common second phase

The IEEE 802.10 working group in their development of [IEEE P802.10C] has taken this direction.

5.18 THE CONCEPT OF SECURITY ASSOCIATION

Key management is responsible for the creation, updating and deletion of security associations. This section is related to the next section 5.19 which explains more in detail the management aspect, traffic encryption key substitution. The term key update is used in this section. As we will see in the next section, traffic encryption keys can be substituted by employing different methods. Key Update is one of the methods described

The notion of a security association originates from SDNS and was originally called a cryptographic association. The article Architectural Model Of The SDNS Key Management Protocol [LAMB88] gives the following explanation of cryptographic association: "A cryptographic association consists of a Traffic Encryption Key (TEK) and associated attributes that determine the usage of the TEK." The cryptographic association is initiated by the key manager. When the key management application association is closed the cryptographic association survives and continues to exist independently. When TLSP is used on a per (transport) connection basis the cryptographic association applies to only a single connection and may not be reused on another concurrent or subsequent connection. The key manager may either immediately end the cryptographic association when the connection closes, or allow it to expire at the end of its cryptoperiod. When TLSP is used on a per NSAP basis, the cryptographic association applies to multiple (transport) connections, both concurrent and consecutive. The key manager may either allow the cryptographic association to expire at the end of its cryptoperiod, or update the traffic encryption key(s) for the cryptographic association and thereby obtain a new cryptoperiod to extend the cryptographic association.

Originally, a security protocol entity identified a cryptographic association by using a key identifier (KID). The value of the KID is established by the key manager. In its simplest form, the KID could be an index into a table where the associated attributes are stored. When receiving a security protected PDU, the KID is extracted from the PDU. The KID is then used as index when retrieving the attributes from the table. By applying the right attributes the PDU could be properly processed. A problem with the described scheme is the duration of the cryptoperiod. In SDNS key management the cryptoperiod is not negotiated. Therefore, one of two communicating parties could have a longer cryptoperiod than the other. The party with the longer cryptoperiod could be sending PDUs which could not be processed on the other end, since the cryptographic association cannot be recognized any more. In particular, this could happen if the two communicating security protocol entities belong to different security domains. [SDNS 90-4262] provides the Kurq (Key Update Request) and Kurs (Key Update Response) PDUs to solve the problem. The one with the shorter cryptoperiod can initiate a Key Update. It seems that for a very short period of time, before the key manager informs the security protocol (SP) entity that the KID has been changed, it would be possible to use a KID that the peer has already changed. However, there is a cryptofacility at the system, and when the SP entity wants to perform an encryption with a key associated with a certain KID, the cryptofacility could inform the SP-entity that the KID is not longer valid. The cryptofacility can act in the following way:

1. The key manager informs the cryptofacility that a key update has to take place. The Key Update request is received from the peer key manager.
2. The cryptofacility transforms the existing traffic encryption keys and returns the KID associated with the new keys. The new KID can be referenced by the key manager.
3. The SP entity wants to perform an encryption using the old KID. The SP entity does not know that the remote key manager has initiated a key update.
4. The cryptofacility returns an error message (maybe indicating key no longer valid).
5. The SP entity requests a replacement KID from the key manager.
6. The key manager informs the SP entity about the new KID.

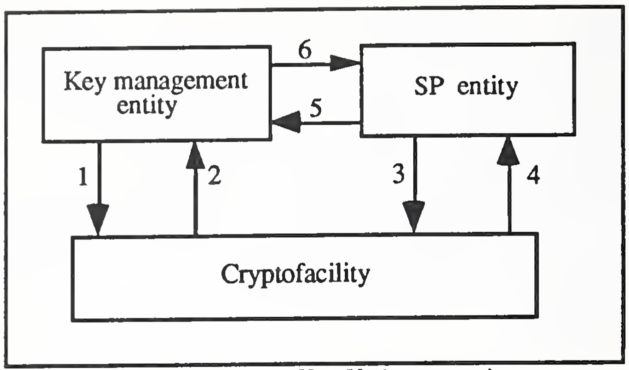


Figure 5.18:1 Key Update scenario

In SDNS the key management entity uses the Nokey PDU to inform its peer that the KID used cannot be recognized by a security protocol entity.

The Lower Security Guidelines document [ISO/SC6 6957] uses a different terminology; the term security association is used instead of cryptographic association. The NLSP standard [ISO 11577] has a security association identifier (SAID) instead of key identifier. [ISO/SC6 6957] gives the following explanation of security association:

> In order to protect an instance of communication (a connectionless SDU or a connection) a collection of information (keys and other attributes needed to control the operation of security) has to be established between the communicating entities. This collection of information is referred to as a security association (SA).

Using the term security association instead of cryptographic association emphasizes the fact that the relationship includes more than just cryptographic keys.

NLSP provides the possibility to establish a security association on-line (it is mentioned earlier that key management is included in NLSP [ISO 11577]). The key management part of [ISO 11577] does not directly address how to perform a key substitution without having to transfer key material. [ISO 11577] mentions that for NLSP-CO, keys can be substituted during the lifetime of a security association. The document is more oriented towards the approach where new key material is transferred than the transformation of existing keys (?).

The current SILS key management draft, [IEEE P802.10C], provides service primitives to update and delete a security association. By updating a security association a new association is created with the same characteristics as a current one. For example, there is no need to negotiate about some of the attributes used by the security protocol. Since [IEEE P802.10C] has not yet reached a mature status, it has not been ultimately decided whether the key material of the updated security association is going to be created from existing key material. However, the current version of [IEEE P802.10C] has followed the approach to derive new keys from existing ones. The intention is to use this feature in conjunction with TLSP. When TLSP is used on a per connection basis the transport sequence number may begin to cycle. In order to resist replay attacks, new traffic encryption keys must be created. For this event, that most likely will occur very rarely, the Update Service Association primitive is provided. In SILS key management if two parties have different cryptoperiods, the party with the shorter period can issue an update of the security association before the period expires. When the new security association is created, the party whose cryptoperiod is going to expire can delete the old security association. The peer can be notified about the deletion through the use of the Delete Security Association service primitive.

There are other views on whether the two entities can have different cryptoperiods. [ISO 11166] mentions that two parties must have the same cryptoperiod. [ISO 11577] assumes the two parties have preestablished An Agreed Set of Security Rules (ASSR) where the cryptoperiod is defined.

5.19 SUBSTITUTION OF TRAFFIC ENCRYPTION KEYS

How should the traffic encryption keys be substituted for a security association? The key manager is responsible for performing the key substitution.

Keys need to be substituted because:

- The cryptoperiod of the keys expires. A cryptoperiod represents, the defined period of time for which a traffic encryption key can be used or, the number of PDUs it is allowed to encrypt/decrypt.
- The transport sequence numbers, used to provide connection integrity when TLSP is used on a per connection basis, may begin to cycle. The traffic encryption keys must be substituted so that TPDUs sent previously in the transport connection cannot be replayed.

Traffic encryption keys may be substituted in three different ways:

- Through the early creation of other traffic encryption keys.
- Through update - the generation of new key material from either the current traffic encryption key or information exchanged previously to form the current TEK.
- Through replacement - the generation of new key material through information contained in further exchanges between key managers.

The first method to substitute the traffic encryption keys avoids communication between the key managers when the keys need to be substituted. Traffic encryption keys are created in advance, during the key substitution procedure a key manager substitutes the traffic encryption keys with other traffic encryption keys already created. The drawback of this solution is the cryptoperiod. The cryptoperiod of the already created traffic encryption keys may expire before they are used. Local security policy determines when the cryptoperiod starts, at the beginning of key usage or at key creation time. Another problem with this method is the ability to store the keys in a secure manner. They must not be compromised before their usage. The advantage of this method is that the key substitution can be performed fast.

The second and third methods are based on the interactions of the key managers when the traffic encryption keys need to be substituted. When the key manager resides in the application layer, one drawback is that many associations which are time consuming to create must be established.

The idea behind the second key substitution method is to perform an efficient key substitution by updating the traffic encryption keys. In order to achieve this, existing key material may be used to create the new traffic encryption keys by applying a function:

new-key = key-update-function (old-key).

In a commercial environment where no assumptions can be made about the public exposure of the algorithms, this procedure has a drawback. If the existing keys are compromised and the update algorithm is known, the future keys are also compromised. Another solution is to reemploy unused portions of the key material. For instance, a Diffie-Hellman exchange creates a long string where only parts of the string are needed for the initial key material. When a key update occurs unused parts of the string could be taken.

When the third method is employed, new traffic encryption key material is created. The new key material is independent of the previous one. For example, new key material can be created by doing a Diffie-Hellman key exchange. The old traffic encryption keys are substituted with new ones created by the key exchange. It is time consuming, especially if it is done in software. What impact will this procedure have on the connections established where the security protocol resides? A transport protocol [ISO 8073] entity can send Acknowledgement TPDUs in order to keep a transport connection open while no DT PDUs are sent. Note that this is listed as a threat in the article Security Mechanisms In A Transport Layer Protocol [VOYD84]. By sending AK-TPDUs, the inactivity timer is prevented from expiring, so it should be possible to do a key replacement and still keep the associations of the security protocol open. The threat described in [VOYD84] is that an intruder can record two AK TPDUs sent in each direction. Thereafter the intruder can discard all TPDUs and play back the recorded AK TPDUs. Thus, the inactivity timer is prevented from expiring. Another solution that has been mentioned when TLSP is used, is to close the transport connection and open a new connection when the key replacement or key updating is done. If the transport connection has to be closed and reopened, which entity has the responsibility to perform that? If it is the transport user, responsibility lies with the session layer. The question is how to inform session that the connection should be closed and after key substitution has occurred reopened. Another question is whether this procedure is prohibited by the Security Architecture 7498/2 [ISO 7498-2], which states that no security services can be placed in the session layer.

When TLSP is used on a per connection basis, transport sequence number begins to cycle at 0 (see Figure 5.19:1), it is in theory possible to perform a replay. If TLSP relies entirely on transport and therefore does no sequence number check, before 0 is reached, a previously sent PDU can be replayed. The replayed PDU must have a sequence number that is greater or equal to 0 and less or equal to n+x.

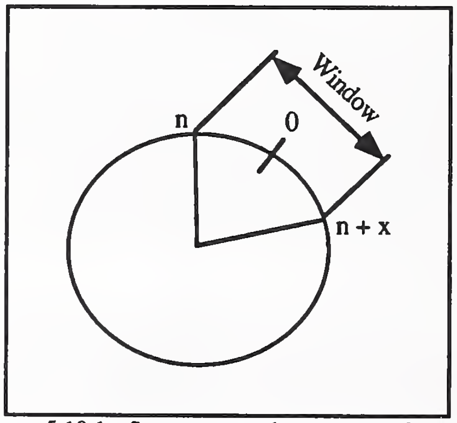


Figure 5.19:1 Sequence number space cycles at 0

The receiving TLSP entity does no sequence number checking and passes the TPDU to transport. Later when the sending TLSP entity has to encapsulate a TPDU with sequence number 0, it discovers that a key substitution must take place. However, the replayed PDU will already be held in the buffer of the receiving transport entity. When the correct TPDU arrives it is discarded by transport. Therefore, when in this situation TLSP should check sequence numbers and drop TPDUs between 0 and n+x until a key substitution occurs.

5.20 KNOWN PLAIN TEXT ATTACK

The transfer syntax representation of ASN.1 [ISO 8825] consists of type, length, and value fields. It is possible to perform a known plaintext attack because the type field contains known values in an encrypted portion of the data. If we study the following [SDNS 90-4262] protocol specifications:

```
Kpdu ::= CHOICE {
   [0] IMPLICIT     Nkrq,
   [1] IMPLICIT     Nkrs,
   [2] IMPLICIT     Tryme,
   [3] IMPLICIT     Nokey,
   [4] IMPLICIT     EKpdu}
EKpdu ::= SEQUENCE {
   remote-kid KeyId,
   contents OCTET STRING -- ENCRYPTED PlainTextKpdu}
PlainTextKpdu ::= SEQUENCE {
   plaintext CHOICE {
               [0]   IMPLICIT    Ssrq,
               [1]   IMPLICIT    Ssrs,
               [2]   IMPLICIT    Estat,
               [3]   IMPLICIT    Kurq,
               [4]   IMPLICIT    Kurs,
               [5]   IMPLICIT    Rkrq,
               [6]   IMPLICIT    Rkrs,
               [7]   IMPLICIT    Srkrq,
               [8]   IMPLICIT    Srkrs,
               [9]   IMPLICIT    Srdrq,
               [10]  IMPLICIT    Srdrs,
               [11]  IMPLICIT    Crrq,
               [12]  IMPLICIT    Crrs},
   padding OCTET STRING OPTIONAL,
   icv IntegrityCheckValue}
```

Note that the application of the ASN.1 encoding rules cause a number of predeterminable type codes to be inserted throughout the *PlainTextKpdu*. For example, the attacker knows that the first encrypted data is the SEQUENCE type identifier of the PlainTextKpdu. In general, the cryptographic algorithm selected must be capable of resisting known plain text attacks.

The content fields within an NLSP Secure Data Transfer PDU do not appear in any special order, as illustrated below:

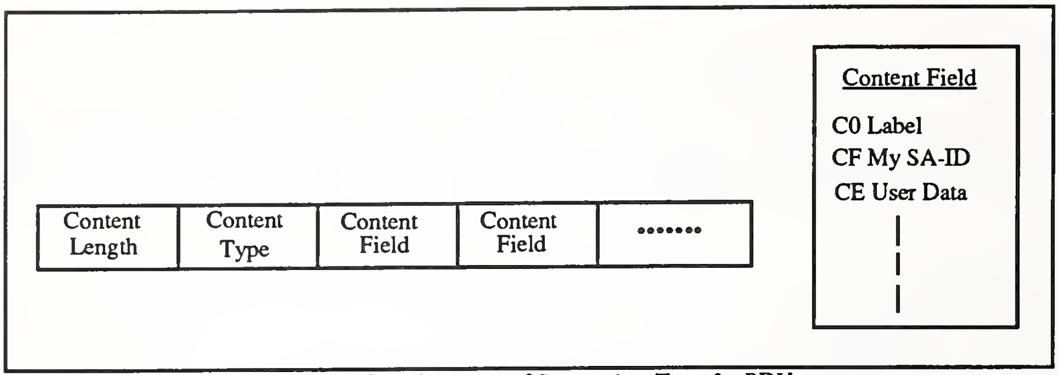


Figure 5.20:1 Structure of Secure data Transfer PDU

By allowing them to appear in any order it is more difficult to predict which field contains particular data. It should, however, be mentioned that this PDU structure was probably adopted for other reasons.

The problems discussed in this section are probably to regard as minor problems.

5.21 ADDRESSING PROBLEMS

The addressing problems occur when two end systems want to communicate securely and the key manager for one or both end systems is not residing in the end system. In some way the initiating key manager has to find out the presentation address of the remote key manager. This is a violation of OSI principles, because an application protocol, in this case the key management protocol, should not be aware of the underlying network.

5.21.1 Addressing problems in SDNS

In SDNS, a gateway can provide the security services for a number of hosts. When a key exchange occurs, the initiating key manager which resides in a gateway, knows the address of the remote host but not the address of the remote key manager. For example, in Figure 5.21:1 (taken from [LAMB88b]), host H3 may want to communicate securely with host H6. The security services are provided by gateways, it is assumed that S4 and S5 are security devices residing in gateways. Probably, because the boxes of H3 and S2 are apart, S2 also resides in a gateway. The available information for S2 when a security association has to be established, is the address of the remote host, i.e., host H6. The presentation address of the key management entity residing in a gateway and serving the host has to be derived from this information.

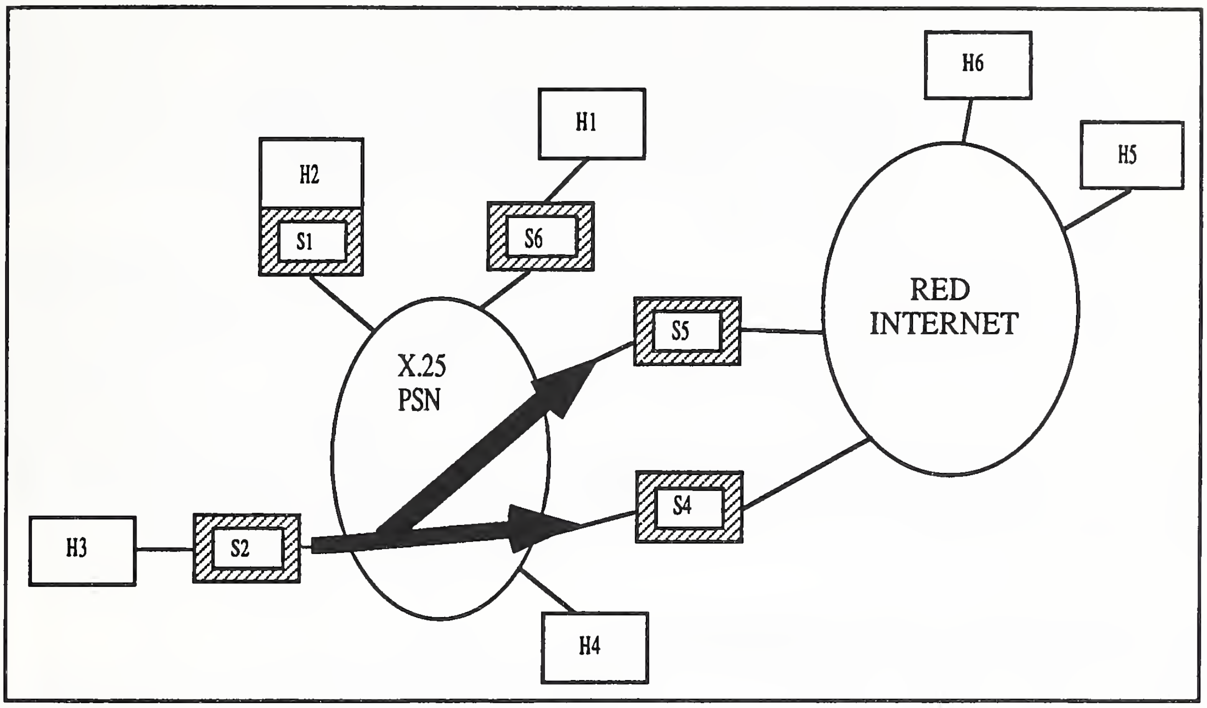


Figure 5.21.1:1 A single path has to be picked

The Tryme PDU of [SDNS 90-4262] is designated for finding out the address of the remote key manager. It is not obvious how the address of the remote key manager is found by using the Tryme PDU. Every OSI application protocol[14] and therefore also a key management application protocol, has to use the Application Control Service Element (ACSE) to establish an application association. In order to establish an application association, the remote presentation address must be known. How, in [SDNS 90-4262] can an application association be established, if the presentation address of the remote entity is unknown? According to [SDNS 90-4262], an application association must be established before the Tryme PDU can be sent.

The documents included in [SDNS 90-4262] do not give any explanation to why the remote address must be found dynamically. One motive can be there exist multiple paths, as illustrated in Figure 5.21:1, and a single one has to be picked. Instead of the Tryme PDU a directory service could be used. The directory service would map from the address or identity of a host to the presentation address of the key manager serving the host.

[14] Here too, only connection oriented application protocols are considered.

Address or Id of host	Presentation address of key manager serving the host
<host id>	presentation address of a key management entity located at a gateway, the host identified by <host id> is attached to the red part of the network. At the gateway are the red/black network conversions done.
⋮	⋮

Figure 5.21.1:2 Translation table

Even if a directory service is available, it does not not solve the multiple path problem. A drawback of having mapping tables is that the tables must be updated whenever new hosts are added to or removed from the network. As we will see in the next section further problems arise if mapping tables were to be used.

5.21.2 Addressing problems in SILS

SILS provides the possibility to have the security protocol entity residing in a bridge which then serves a number of end stations. For example, in an environment as shown in Figure 5.21.2:1 the local portion of the LAN may be regarded as secure, this is denoted as Protected Subnet. However, the traffic going out from the local portion of the LAN needs to be protected. A bridge equipped with the SDE protocol [IEEE P802.10B] can protect the traffic going out from the local portion of the LAN.

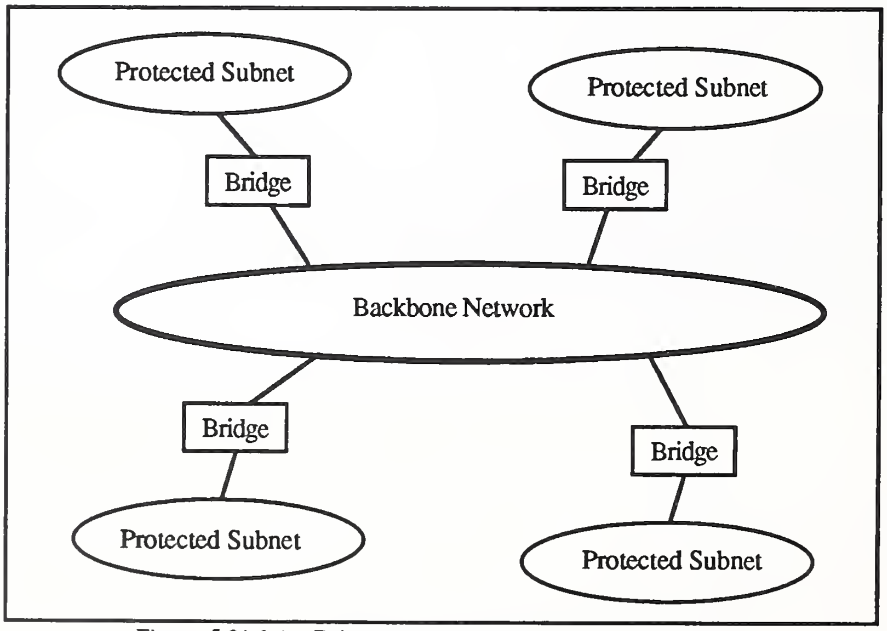


Figure 5.21.2:1 Bridges are providing the security services

The key manager can, thus, be located at a bridge. When a security association has to be established, the address of the remote key manager located at another bridge is unknown, but the medium access control (MAC[15]) address of the remote end station is known. In order to find out the address of the remote key manager, probe packets are sent from layer two. By sending a probe packet from layer two, the problem of establishing an association before knowing the presentation address is avoided.

The motive given in the SILS key management proposal to why not having tables containing the translation from end station addresses to presentation addresses of key managers serving the stations, is that such a data base would be quite large. It may even be difficult to have translation tables because bridges can be assigned dynamically. For example, when the spanning tree algorithm is used, the failure of the bridge can cause traffic to be directed in another direction ([STALL90] provides more details about the spanning tree algorithm). The following figure illustrates such an environment:

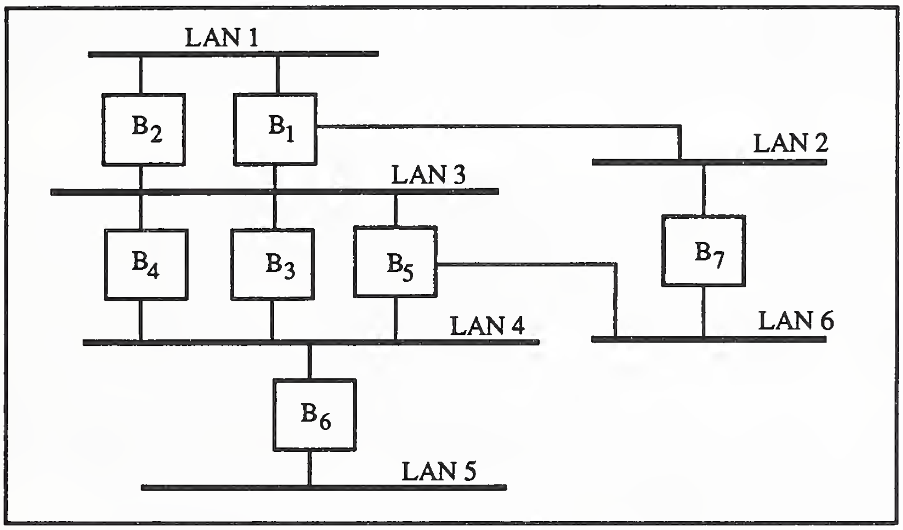


Figure 5.21.2:2 Environment where bridges are assigned dynamically

The example is taken from the article Transporting Bridges For Interconnection Of IEEE 802 LANs [BACK88]. Traffic going from LAN 1 to LAN 6 goes through B_1 and B_5. However, if B_5 were to fail, B_7 would be forwarding traffic to and from LAN 6. In such an environment it may not be possible to predict which bridge will forward the traffic to and from a LAN. If tables are used, possibly there must be several presentation addresses for each end station address. When the key manager opens an application association, it must test each presentation address until an association can be established. The presentation addresses could appear in a priority order. The alternative is to find out the presentation address dynamically.

Parallel bridges present another problem to key management. Figure 5.21.2:3 shows a hypothetical example of how parallel bridges may be configured:

15 Not to be confused with the MAC of [ANSI X9.17] and [ANSI X9.28].

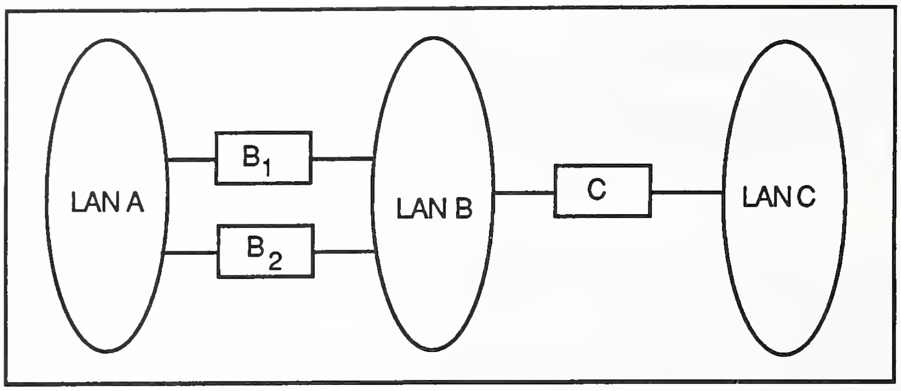


Figure 5.21.2:3 Parallel bridges

If a packet is originating from bridge C, it can be passed to either bridge B_1 or B_2. B_1 could send a request PDU and B_2 might receive the response PDU. Then, key managers located at B_1 and B_2 are not able to operate correctly. There are parallel bridges in Figure 5.21.2:2 too, however, when the spanning tree algorithm is used only one of the parallel bridges forwards the traffic to and from its LAN.

The following example explains in more detail how, when there is no security association established, the presentation address of the remote key manager is found dynamically. The description of [IEEE P802.10C] also includes different kinds of error situations. For example, an error situation may occur when a security association between the systems has existed, but what in the following description is called the initiating system was powered down so the security association was lost. However, what in the following description is called the responding system may still regard the security association as active. In Figure 5.21.2:4 three probe packets are exchanged between the layer two entities, in order for the initiating key management entity to determine the presentation address of the remote key management entity. These probe packets are: PROBE.request, PROBE.response and PROBE.confirm. The presentation address of the remote key manager is found out in the following way:

1. A PROBE.request packet is sent from the initiating entity, included as parameters in the packet are: a PROBE ID so the response packet arriving can be associated, the presentation address of the local (initiating) key manager, the MAC address of the initiating end station and the MAC address of the local (initiating) bridge. The packet is sent to the remote end station which guarantees that the packet is handled by the appropriate bridge.
2. If the responding entity cannot find a security association established with the initiating entity, it creates a KMASO[16] SAID (KSAID) to identify the association between the two KMASOs. The PROBE.response packet includes the KSAID, the PROBE ID, the MAC address of the local (responding) bridge and the presentation address of the local (responding) key manager. The responding bridge will know to which bridge to send the packet because the MAC address of the initiating bridge was included in PROBE.request. If the responding entity finds that a security association has already been established, it will include two KSAIDs in the PROBE.response packet.
3. If only one KSAID is included, it is a confirmation that no security association is established between the entities, the local KSAID value is then created. The local KSAID value is sent in the PROBE.confirm packet to the remote bridge. The MAC address of the responding bridge is known because it was included in the PROBE.response packet.

[16] Application Service Object (ASO) is defined in [ISO 9545/P1].

4. The local Key Management Application Process is informed that keys and attributes have to be established with a key management entity, whose presentation address was found by using the probe packets.
5. The Key Management Application Process invokes the Key Management Application Service Element (KMASE).
6. An application association is established using ACSE.
7. The two key managers establish the keys and attributes.
8. The application association is terminated using ACSE.

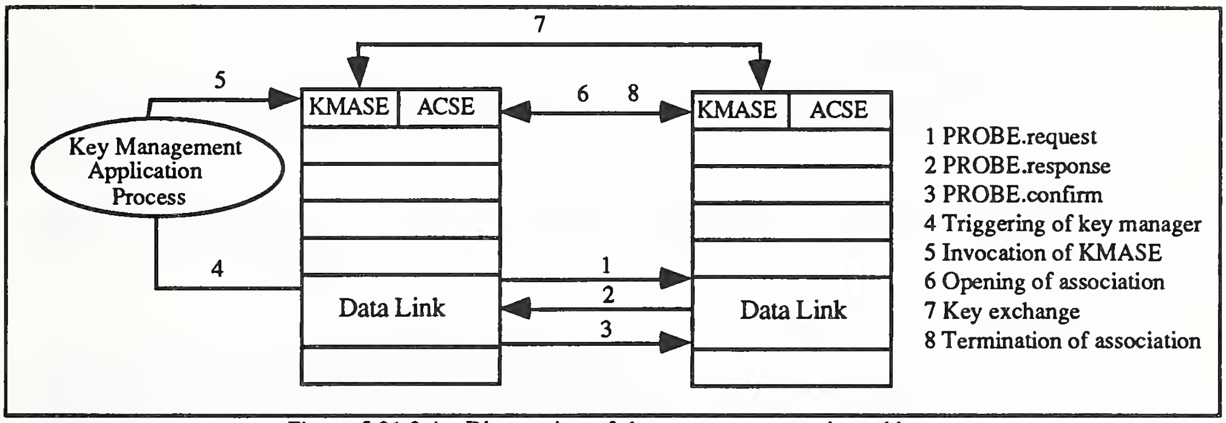


Figure 5.21.2:4 Discovering of the remote presentation address

5.21.3 Can key management entities be trusted?

Are key management entities trusted entities? If they are not trusted, a key management entity can pretend to serve a host which in reality it does not. So the question is whether the sender of a probe packet can be sure that the responding part is the real key manager for a particular host.

One might regard the SDNS world as a single security domain where the different SDNS components trust each other. In such an environment, an SDNS component may not fear to be cheated by another SDNS component since they belong to the same community. It is the outside world that is hostile. Can there be a similar level of trust in an SILS environment? SILS is not designed for a particular security domain as SDNS is; the scope of SILS is to be applicable for a wide range of environments. For example, according to Figure 5.21.2:1 SILS could be used in an environment where the different subnets may be managed by different security domains. Some of these security domains may contain or represent competitors that would not trust a key management entity claiming to serve a particular host. It can be a competitor that wants to gain possession of information of another competitor. In such an environment, if there still is a desire to find out the remote address dynamically, authentication of the application entities protected by the security association (e.g., FTAM) becomes essential. For example, authentication in ACSE [ISO 8649/AD1] [ISO 8650/AD2] could verify that communication is established with the right application entity. This is similar to the situation in [ANSI X9.17] where one party having a keying relationship with a key center could masquerade as another such party if notarization was not employed.

5.22 ASPECTS OF KEY MANAGEMENT OUTSIDE OSI

In this section a number of issues are collected which are outside the scope of OSI.

5.22.1 Key Archiving

The Security Architecture document [ISO 7498-2] considers key archiving not to be part of the OSI aspects of key management. [ISO 11166][17] mentions that all archived keys shall be enciphered under a key designated for that purpose.

5.22.2 Key Generation

[ISO 7498-2] does not address this issue, so it cannot be regarded to be part of the OSI aspects of key management.

Often, key management standard documents state the requirements on the key generation procedure, without specifying a particular key generation method. For example, in [ISO 11166] it is written:

> The generation of symmetric keys and initialisation [initialization] vectors shall be by means of a process that ensures that all keys and initialisation [initialization] vectors are random or pseudo-random. The design of this generation process shall be such that no cryptographic advantage is gained by attacking the key generation process rather than the encipherment process.

Appendix C of [ANSI X9.17] is an exception, it gives an example of how DES keys can be generated by using a pseudo-random process. An often cited article in this area is Generation, Distribution, And Installation Of Cryptographic Keys [MATY78].

[ISO 11166] also places demands on the key generation process of asymmetric keys. It states that the generation of the asymmetric key set shall employ a process that ensures the random or pseudo-random process from at least 10^{40} possible values. Another part of the standard, that is, [ISO 11166-2], addresses this issue further. An often cited article in this area is Strong RSA Keys [GORD 84].

5.22.3 Key Destruction

[ISO 7498-2] does not address this issue. [ISO 11166] mentions that plaintext keys shall be zeroized at the end of their life. Zeroization is defined as: "A method of erasing or overwriting electronically stored data." The standard also addresses how to destroy printed keys.

Appendix E of [ANSI X9.17] explains how keys should be zeroized when they are stored in different types of medias.

[17] As mentioned in section 1.3.4, [ISO 11166] is not intended for an OSI environment.

5.22.4 Key Storage

Key Storage is not addressed in [ISO 7498-2]. The storage of keys is related to the cryptographic facility because keys primarily reside in the cryptographic facility. If the keys are stored outside the cryptographic facility, they must not be available in plaintext form. For example, if traffic encryption keys are created in advance and there is not enough memory in the cryptofacility, the keys have to be stored on a secondary media encrypted with a masterkey. Key management standard documents often mention the requirements a cryptographic facility should fulfill. [ANSI X9.17] contains the following phrase: "The key management facility shall be designed so as to protect its contents from unauthorized disclosure, modification, substitution, insertion, and deletion."

[FIPS 140-1] specifies the requirements for a cryptographic facility to be used by U.S. Government Agencies. This draft standard specifies four different levels of security, where level four is the most advanced. As mentioned earlier in this paper, the Draft FIPS proposal for key management using ANSI X9.17 [FIPS 171] stipulates the cryptofacility to have level three functionality.

The book Cryptography: A New Dimension In Computer Security [MEYE82] provides an explanation of a cryptofacility.

5.22.5 Compromised Key Recovery

[ISO 7498-2] does not address this issue. [ISO 11166] mentions that if the secret[18] key of the CKC is suspected to be compromised, the CKC shall take immediate action to generate a new key set and distribute the new CKC public key.

[RFC 1114] points out that the compromise of a CA´s private key component leads to the revocation of all subordinate certificates.

18 The term private key is used in this paper.

6 REALIZATION OF THE MODEL

How can the model presented in chapter 3 be realized? Of the key management schemes reviewed in chapter 5, some of the possibilities to realize the model are:

- In the case of an asymmetric form of key management the Key Center object can be realized by the certification authority scheme as described in the Directory Authentication Framework [ISO 9594-8] or as described in [RFC 1114].

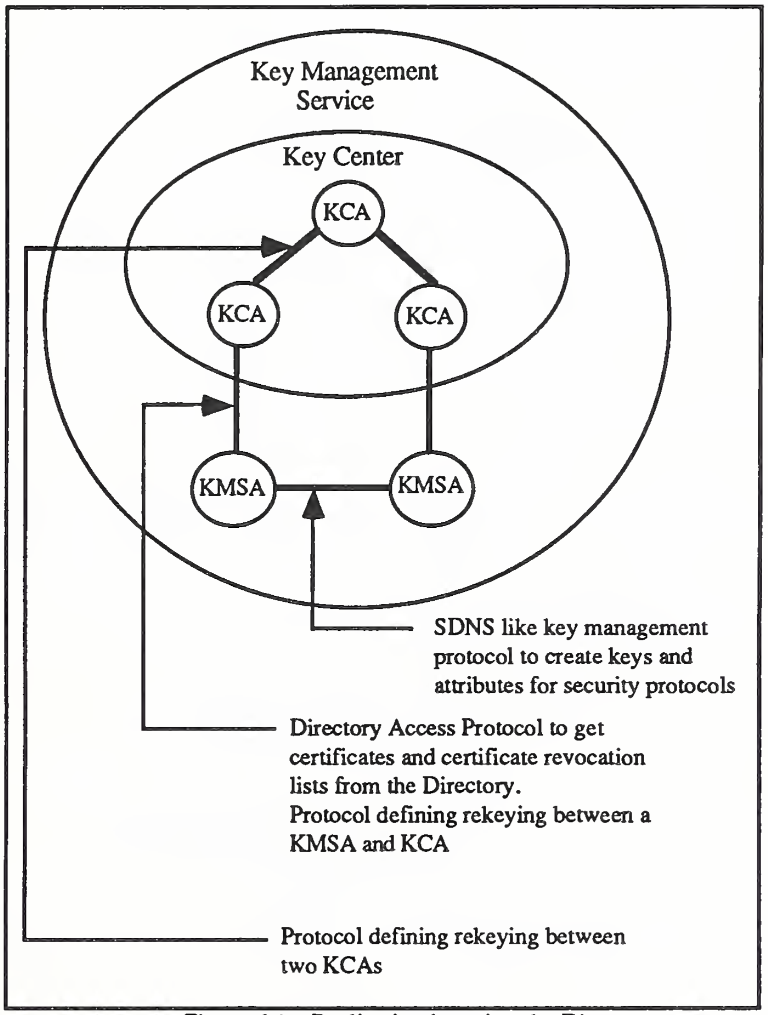

Key Management Service
Key Center
KCA
KCA
KCA
KMSA
KMSA
SDNS like key management protocol to create keys and attributes for security protocols
Directory Access Protocol to get certificates and certificate revocation lists from the Directory.
Protocol defining rekeying between a KMSA and KCA
Protocol defining rekeying between two KCAs

Figure 6:1 Realization by using the Directory

An SDNS-like key management protocol can be used to create the keys and attributes needed by the security protocols. This protocol specifies the KMSA-KMSA interactions. A KMSA interacts with a KCA to get certificates and certificate revocation lists. This could be described by the Directory Access Protocol [ISO 9594-5]. The KMSA is then acting as a Directory user and accesses the Directory through the Directory User Agent. Two protocols are also needed to make it possible to perform rekeying between KMSA-KCA and KCA-KCA. This structure is consistent with the ongoing activities in IEEE 802.10 to develop the asymmetric part of the key management protocol.

- The Key Center object can for a symmetric form of key management be realized by an [ANSI X9.17] Key Distribution Center or Key Translation Center and the KMSAs by the [ANSI X9.17] parties.

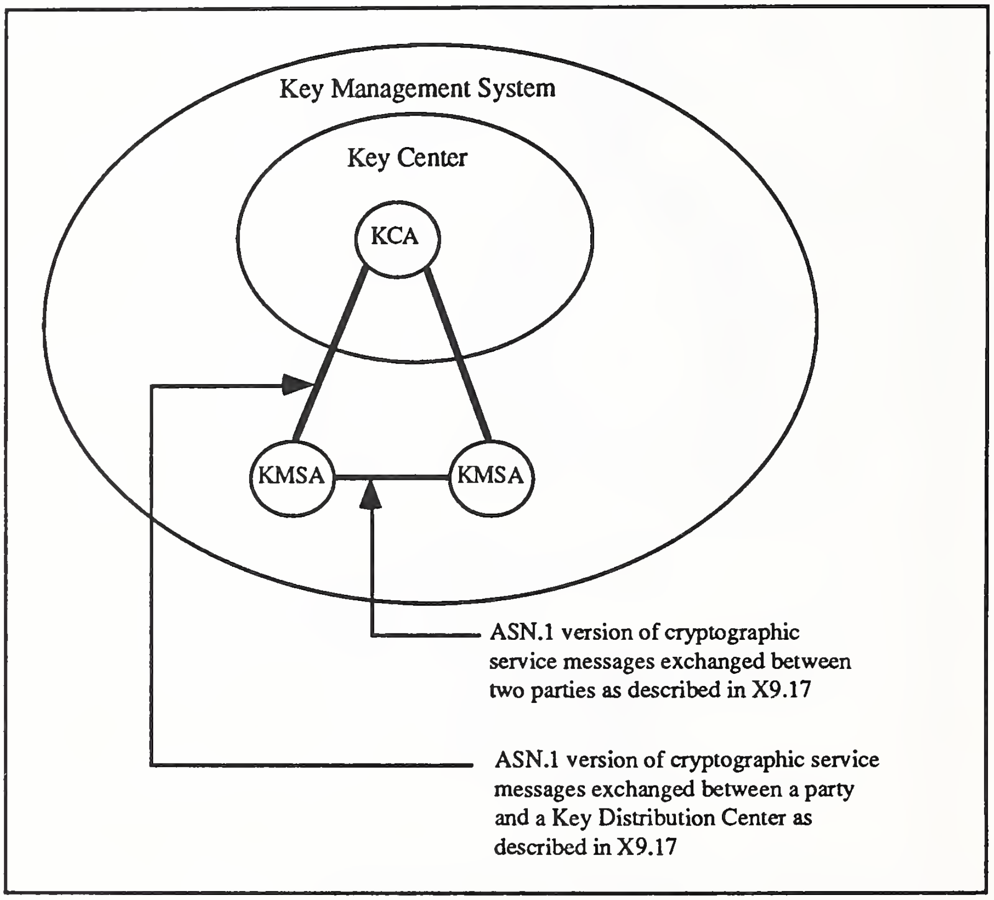

Key Management System
Key Center
KCA
KMSA
KMSA
ASN.1 version of cryptographic service messages exchanged between two parties as described in X9.17
ASN.1 version of cryptographic service messages exchanged between a party and a Key Distribution Center as described in X9.17

Figure 6:2 Realization by using X9.17

The protocols defining the interactions between two KMSAs and KMSA-KCA would be ASN.1 versions of the cryptographic service messages defined in [ANSI X9.17]. However, the protocol between two KMSAs must be enhanced because [ANSI X9.17] is not aware of OSI security protocols. The service negotiation phase as discussed in chapter 4 must be added to [ANSI X9.17]. This structure is consistent with the symmetric part of the IEEE 802.10 Key Management protocol.

- If there is a need of multiple centers, the Key Center object can be realized by an [ANSI X9.28] Multiple Center Group.

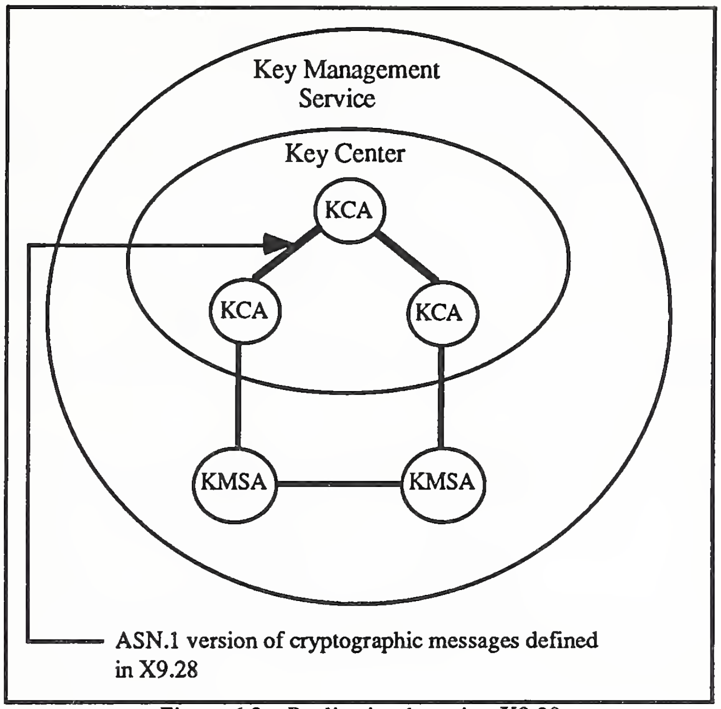


Figure 6:3 Realization by using X9.28

The protocol between two KCAs would be an ASN.1 version of the cryptographic service messages defined in the [ANSI X9.28]. The protocols between two KMSAs and KMSA-KCA can still be the ASN.1 version of [ANSI X9.17].

- Alternatively, existing schemes could be discarded and a new infrastructure could be created.

7 GENERALITY OF THE KEY MANAGEMENT MODEL

The idea behind the key management model presented in chapter 3 is to have a general key management model based on application layer concepts. The model was not able to fully describe all of the key management schemes reviewed in chapter 5. The restrictions are:

1. [ANSI X9.28] messages can flow in different routes, the model cannot, however, describe the flow in different routes. The KCAs of the model are hierarchically related to each other which may not be the case for [ANSI X9.28] key centers.

2. SILS [IEEE P802.10B] [IEEE P802.10C] secret keys can be shared between several entities, the model is oriented towards the approach where secret keys are shared between two entities.

3. The model assumes that a KCA resides within the juristdiction of a specific security domain, and the CA functionality resides within a KCA. The relationship between a certification authority and a security domain may not be so strong. For example, if a public notary acts as a certification authority, the only thing the notary may do is to vouch for the binding between the distinguished name of a user and the public key component. It might therefore be possible to have two users, who have their certificates signed by the same certification authority, but belonging to different security domains.

4. When CRLs and certificates are obtained from the Directory, the relationship may not be as implied by the model. For example, a KMSA can establish an application association with a KCA higher in the hierarchy than the immediate superior KCA.

The problems encountered are probably common problems when modelling is done. If a model is very general, it does not describe very much. On the other hand, when a model is very specific existing schemes may be precluded.

7.1 IMPROVEMENTS OF THE MODEL

This section is an attempt to improvement of the defined key management model, therefore some of the difficulties mentioned previously can be eliminated.

Part 2 of the Directory standards, Models [ISO 9594-2], contains models describing different aspects of the Directory. For example, section 2 of [ISO 9594-2] contains a model describing how a Directory user obtains Directory services. Section 2 also contains a model which describes the composition of DSAs. Both models describe OSI communication aspects since DUAs and DSAs are manifested as application processes. Section 3 of [ISO 9594-2] contains models describing other aspects of the Directory which are not part of the communication aspects. Among other things, it describes the organization of the Directory Information Tree (DIT).

The key management model should have a similar structure to that of the Directory. Now the same model describes both communication and logical aspects; instead those aspects should be represented by distinct models. For example, the decomposition of the Key Center object is done in the following way:

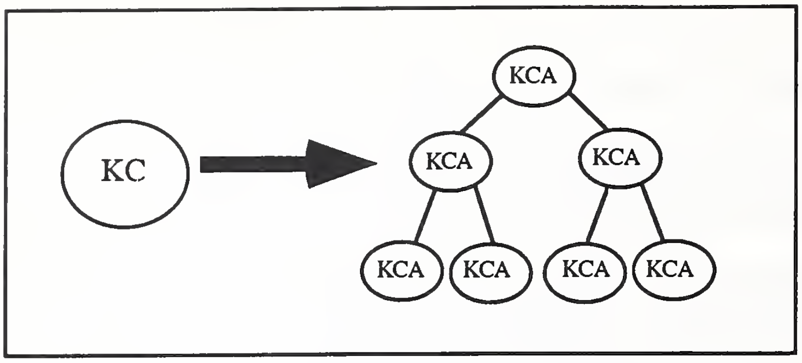


Figure 7.1:1 Decomposition of the Key Center object

In the right part of the above figure both communication and logical aspects are described. The communication aspects are concerned with which protocols are used for the communication between two KCAs. The logical aspect is that certification authorities are hierarchically related, which is not part of the communication aspects. The following figure is right from a logical point of view but not from a communication point of view:

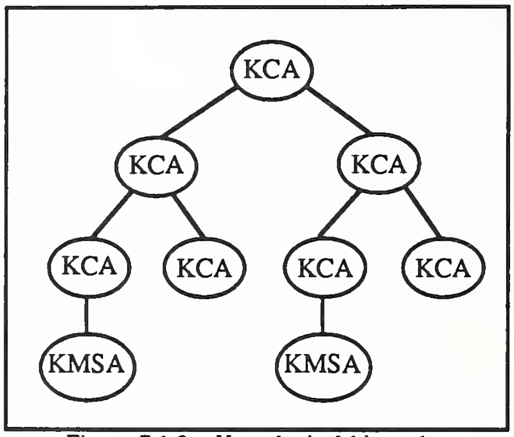


Figure 7.1:2 Hypothetical hierarchy

What cannot be expressed by the above figure is that a KMSA can access a KCA higher in hierarchy than its immediate superior KCA. A KMSA can access such a KCA in order to get a certificate revocation list. Therefore, it would have been better to separate the communication and logical aspects. The communication aspects could have been describes in the same way as in [ISO 9594-2]. The way certification authorities are related could have been described in a separate model. The model concerning the communication aspects can probably describe both an asymmetric and symmetric form of key management. Presumably there must exist different models describing the logical aspects.

8 LEVEL OF CENTRALIZATION

The different key management schemes reviewed in chapter 5 describe different levels of centralization.

The symmetric form of key management as described in the [ANSI X9.17] Key Distribution Center Environment, includes a centralized trusted third party which must be involved in every key exchange.

The SDNS model includes a centralized trusted place, the Key Management Center, but the Key Management Center is not involved when two SDNS components perform a key exchange. This is a benefit compared to the [ANSI X9.17] CKD environment; the overhead associated with a center being involved in every key exchange is avoided. This minimizes risk that the KMC becomes a bottleneck in the network.

The asymmetric form of key management described in [IEEE P802.10C] is the least centralized; it calls for distributed key management. The central place functionality is replaced by the Directory Authentication Framework scheme where there is a hierarchy of certification authorities.

The distribution of responsibility varies between the different schemes. The more centralized the system is, the less responsibility the user of the key management service has. The concept of responsibility also applies to key generation. For a centralized system the Key Distribution Center is responsible for the key generation, the user is not given this responsibility. Whereas in a distributed environment the user is given the responsibility. This is described further in [DSSD87].

9 KEY MANAGEMENT PROTOCOL SPECIFICATIONS

This chapter contains a description of the key management protocol included in Annex B. The protocol is defined as a layer seven Application Service Element (ASE) and maps its PDUs onto the Association Control Service Element (ACSE) and the presentation layer. If a ROSE based ASE is preferred, it should not be too difficult to redefine the protocol in accordance with the ROSE definitions. Macros are often criticized for being hard to implement; most of the macros included in the protocol definitions of this chapter can be substituted by ordinary ASN.1 data types. Unfortunately, it is very difficult to achieve algorithm independence without using macros. Therefore, if macros are used to achieve algorithm independence, then macros representing other ASN.1 data types can also be used to clarify the nature of an operation.

The intention is not to provide the ultimate solution to key management, but to identify the problems involved with defining a layer seven key management ASE. For instance, the security register is just hypothetical. It is used to illustrate the ideas of how the algorithm independence problem can be solved, but only the ASN.1 definitions and associated problems are explored. Also note that no efforts have been made to make efficient ASN.1 definitions. Probably some of the ASN.1 definitions could have been tagged better. A lot of the definitions are declared as optional in order to achieve flexibility, however, a reduced flexibility would facilitate an implementation of the protocol. In some places alternative definitions are provided. It is felt that by including the flexibility and by providing alternative definitions, problems involved in specifying a layer seven key management protocol are highlighted.

There are no declarations made concerning the certificate revocation list. It is assumed that such a list can be obtained from a directory service. The protocol does not address how the certificates can be obtained, here again assistance from a directory service is assumed.

9.1 TYPES OF PROTOCOLS

There are three types of protocols:

- KMSA to KMSA protocol describing the interactions between two KMSAs.
- KMSA to KCA protocol describing the interactions between a KMSA and a KCA.
- KCA to KCA protocol describing the interactions between two KCAs.

By dividing them into three different protocol types a higher degree of flexibility can be achieved. For example, a security domain using an asymmetric form of key management can decide to do the rekeying manually. Thus, only the KMSA-KMSA protocol will be used.

The protocol specifications consider only the asymmetric form of key management, but it is possible to include declarations for a symmetric form of key management. The approach chosen is according to the asymmetric alternative described in chapter 6.

9.2 ASN.1 MODULE DISPOSITION

The ASN.1 modules consist of:

- A key management module (KMSA-KMSA, KMSA-KCA or KCA-KCA) containing the algorithm independent ASN.1 definitions.
- A security register module containing the algorithm dependent definitions.

Figure 9.2:1 illustrates the ASN.1 modules disposition:

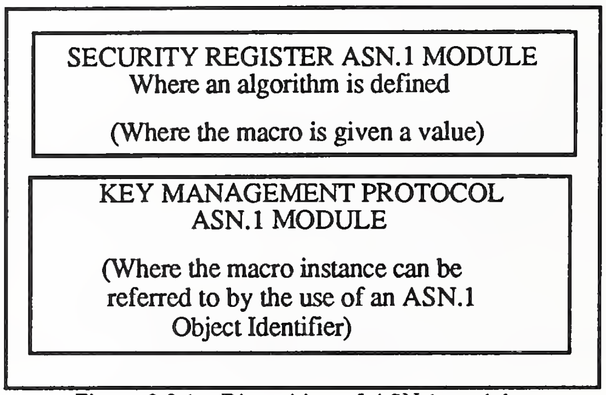

SECURITY REGISTER ASN.1 MODULE
Where an algorithm is defined
(Where the macro is given a value)
KEY MANAGEMENT PROTOCOL ASN.1 MODULE
(Where the macro instance can be referred to by the use of an ASN.1 Object Identifier)

Figure 9.2:1 Disposition of ASN.1 modules

The security register and key management protocol modules import the generic definitions appearing in the Security Definitions ASN.1 module to avoid redefinitions.

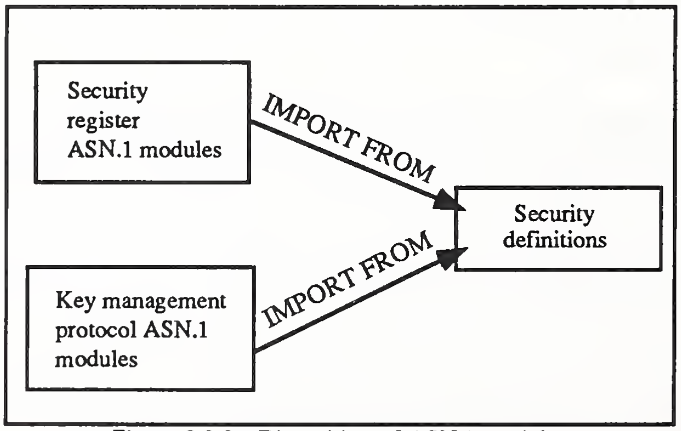

Security register ASN.1 modules
IMPORT FROM
Security definitions
IMPORT FROM
Key management protocol ASN.1 modules

Figure 9.2:2 Disposition of ASN.1 modules

The ASN.1 definitions which are common for the KMSA-KMSA, KMSA-KCA and KCA-KCA protocol modules are defined in the Key Management protocol module.

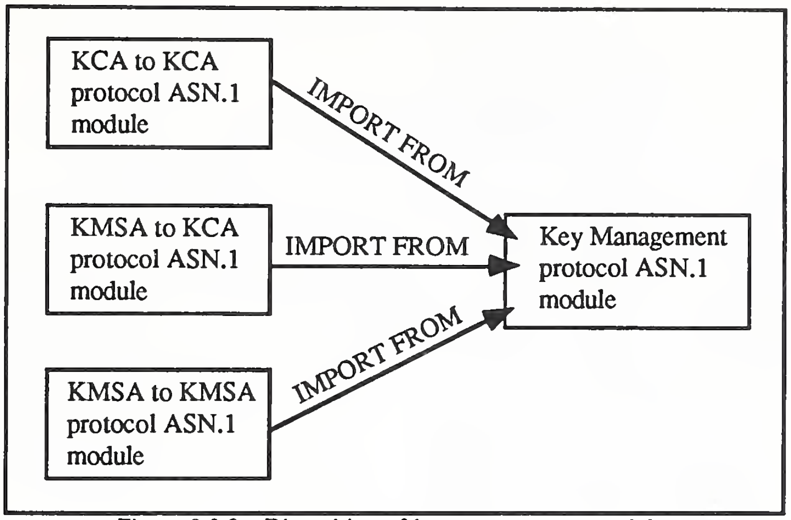

KCA to KCA protocol ASN.1 module
KMSA to KCA protocol ASN.1 module
KMSA to KMSA protocol ASN.1 module
IMPORT FROM
IMPORT FROM
IMPORT FROM
Key Management protocol ASN.1 module

Figure 9.2:3 Disposition of key management modules

9.3 SECURITY REGISTER

The security register contains, among other things, the algorithm dependent data associated with a specific key exchange algorithm. Examples of objects to be included in a security register are:

- Key exchange algorithms/methods;
- Confidentiality algorithms;
- Integrity algorithms;
- Security Labels format;
- Security levels representation;
- Rekeying methods;
- Signature algorithms.

Each of the above mentioned security objects are given a position in the Object Identifier tree. An ASN.1 Object Identifier value is given to the security register itself; this value is represented in the security register module as *{security}*. The dispositions of the objects within the security register are defined by the following ASN.1 definitions:

```
keyExchMethods      OBJECT IDENTIFIER     ::= {security 1}
integrityAlgs       OBJECT IDENTIFIER     ::= {security 2}
confidentialityAlgs OBJECT IDENTIFIER     ::= {security 3}
securityLevels      OBJECT IDENTIFIER     ::= {security 4}
securityLabels      OBJECT IDENTIFIER     ::= {security 5}
signatureAlgs       OBJECT IDENTIFIER     ::= {security 6}
rekeyingMethods     OBJECT IDENTIFIER     ::= {security 7}
```

The hierarchical relationship is:

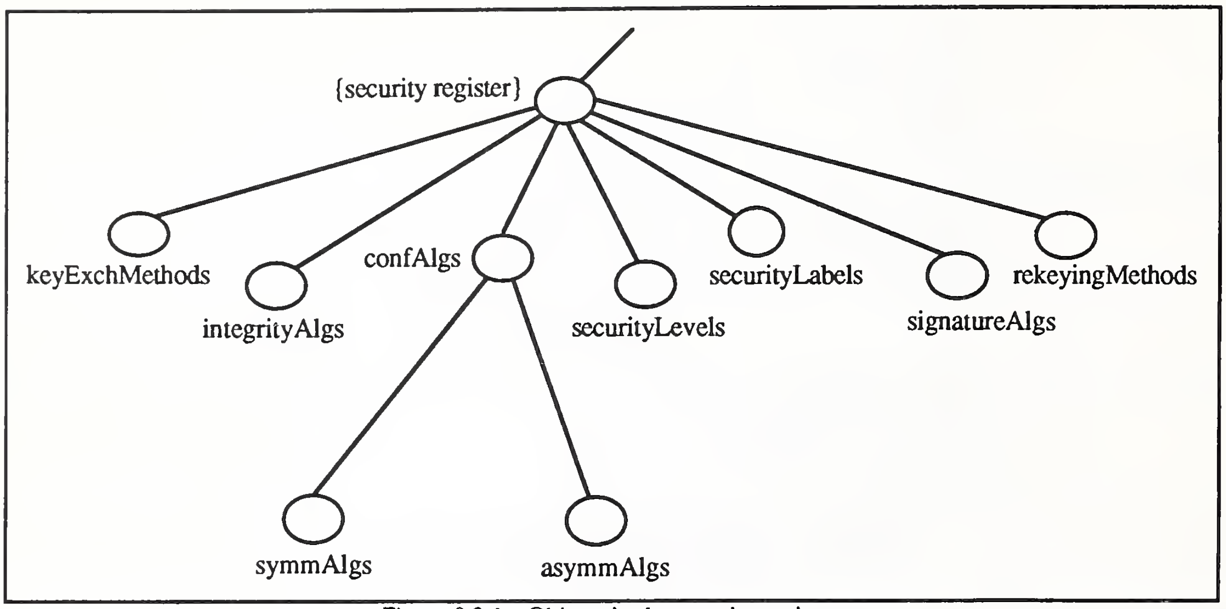


Figure 9.3:1 Objects in the security register

A security register could be either publicly or privately defined. If the security register is privately defined, the content of the security register may not be known to others than the holder of the register.

9.3.1 Registration of key exchange algorithms/methods

The term "key exchange method" is used because there could be several registrations of basically the same algorithm. For Diffie-Hellman there are different ways to perform the authentication, and each particular way is registered separately. Thus, the Diffie-Hellman algorithm can be represented in a security register in different ways.

Section 5.10 discusses the fact that algorithm independence can be achieved by using ASN.1 Object Identifiers in combination with the ANY DEFINED BY construct or by changing presentation context. In the key management protocol specifications the first approach is chosen. That is, a key exchange is described as a four-way handshake procedure. For the KMSA-KMSA protocol the data associated with a particular key exchange algorithm/method is carried by the Nkrq, Nkrs, Ssrq and Ssrs PDUs. The place-holders of the Nkrq, Nkrs, Ssrq and Ssrs PDUs are filled with the algorithm dependent data. The algorithm dependent data is defined in a security register. An Object Identifier functions as a pointer to a certain entry of a security register. The entry defines the data which is associated with the registered key exchange method.

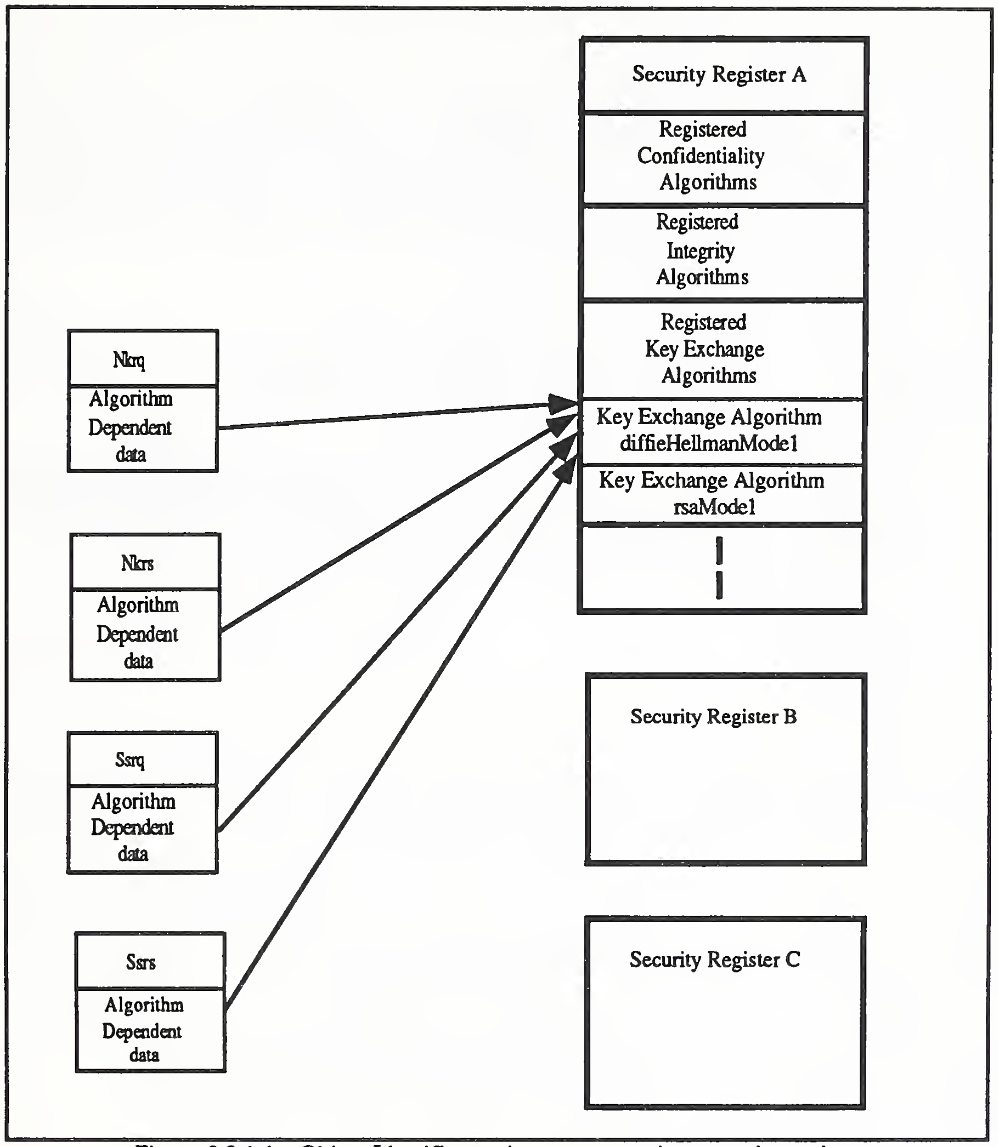

Security Register A
Registered Confidentiality Algorithms
Registered Integrity Algorithms
Registered Key Exchange Algorithms
Key Exchange Algorithm diffieHellmanModel
Key Exchange Algorithm rsaModel
Nkrq
Algorithm Dependent data
Nkrs
Algorithm Dependent data
Ssrq
Algorithm Dependent data
Ssrs
Algorithm Dependent data
Security Register B
Security Register C

Figure 9.3.1:1 Object Identifiers point to an entry in a security register

Two different solutions are proposed, the first uses ASN.1 macros to register each algorithm dependent argument included in a key exchange with a unique Object Identifier. The algorithm dependent data in the Nkrq PDU is associated with an Object Identifier, the data in the Nkrs PDU is associated with another Object Identifier, etc. The second method uses a macro to register all arguments with a single Object Identifier. The algorithm dependent data appearing in the Nkrq, Nkrs, Ssrq, Ssrs, Rkrq, Rkrs, Rkdrq and Rkdrs PDUs are associated with a single Object Identifier. Two different approaches are proposed because the second approach is probably harder to implement, but it has the advantage of only using one Object Identifier. It is harder to implement because the ASN.1 parser must know in the PDU type (context) the algorithm dependent data is conveyed. When each piece of the algorithm dependent data is associated with a unique Object Identifier, the ASN.1 parser does not need to be aware of the PDU type that conveyed the information. It can translate from transfer to abstract syntax by just knowing the value of the Object Identifier. This can be a fall-back if it is shown to be difficult to implement the macro associated with just one Object Identifier.

9.3.1.1 Approach one

The macros needed when the first approach is chosen are ALGORITHM and ALGORITHM-ARGUMENT-REGISTRATION. The macro ALGORITHM is defined in the Directory Authentication Framework [ISO 9594-8] and ALGORITHM-ARGUMENT-REGISTRATION is defined in the KeyExchangeMethods ASN.1 module. The latter macro has the following definition:

```
ALGORITHM-ARGUMENT-REGISTRATION MACRO    ::=
BEGIN
   TYPE NOTATION          ::= "ARGUMENT" type
   VALUE NOTATION         ::= value (VALUE OBJECT IDENTIFIER)
END -- of ALGORITHM-ARGUMENT-REGISTRATION
```

The Diffie-Hellman based algorithm as presented in Key Management Systems Combining X9.17 And Public Key Techniques [GRAFF90], can be registered as a key exchange method by the use of the macro ALGORITHM. See the following ASN.1 declaration:

```
diffieHellmanMode1 ALGORITHM
   PARAMETER SEQUENCE {
                    A,
                    P}
   := {keyExchMethods 1}
```

{keyExchMethods 1} indicates that it is the first branch of the keyExchMethods subtree, it is the first registered key exchange method.

```
A ::= INTEGER

P ::= INTEGER
```

A and P represent the values a and r of the Diffie-Hellman algorithm.

The arguments appearing in Nkrq and Nkrs are defined according to:

```
diffieHellmanMode1NkrqData ALGORITHM-ARGUMENT-REGISTRATION
   ARGUMENT SEQUENCE {
         certPath CertificationPath,
         unidirectKeys BOOLEAN OPTIONAL,
         SIGNED EXPONENTIATION A P RandomA}
   ::= {keyExchMethods 1 1}


diffieHellmanMode1NkrsData ALGORITHM-ARGUMENT-REGISTRATION
   ARGUMENT SEQUENCE {
         certPath CertificationPath,
         unidirectKeys BOOLEAN OPTIONAL,
         SIGNED EXPONENTIATION A P RandomB}
   ::= {keyExchMethods 1 2}

RandomA ::= INTEGER

RandomB ::= INTEGER
```

EXPONENTIATION A P RandomA and *EXPONENTIATION A P RandomB* represent $\alpha^X \bmod \rho$ and $\alpha^Y \bmod \rho$ respectively. *SIGNED EXPONENTIATION A P RandomA* and

SIGNED EXPONENTIATION A P RandomB represent the numbers in conjunction with the signatures computed on the numbers. The macro SIGNED is defined in [ISO 9594-8], the macro EXPONENTIATION is explained later. It is probably not necessary to sign these numbers because they may be so small that they do not need to be hashed. *unidirectKeys* is used to determine whether unidirectional keys are to be used. The algorithm described in [GRAFF90] does not need to have any data present in the Ssrq or Ssrs PDUs. In those PDUs the algorithm dependent data is declared as an optional field.

```
Ssrq ::= SET {
   alg-dep-data [0] IMPLICIT SET {
         algArgId          [0]  IMPLICIT OBJECT IDENTIFIER OPTIONAL,
         algArgument       [1]  IMPLICIT ANY DEFINED BY algArgId} OPTIONAL,
   said [1] IMPLICIT Said,
   spParameters CHOICE {
            [2] IMPLICIT TLSPParams,
            [3] IMPLICIT NLSPParams,
            [4] IMPLICIT SDEParams}}
```

The Ssrs has the same ASN.1 representation as the Ssrq.

EXPONENTIATION is a macro describing the exponentiation in Diffie-Hellman. It has the following definition:

```
EXPONENTIATION MACRO ::=
BEGIN
   TYPE NOTATION          ::= type(Alpha) type(Rho) type(Random)
   VALUE NOTATION         ::= value(VALUE INTEGER)
   -- the value of the INTEGER is generated by exponentiation
   -- of Alpha with Random mod Rho, i.e., (a ** random) mod p
END -- of EXPONENTIATION
```

Random ::= INTEGER

Random, represents a random number, X or Y, in the Diffie-Hellman algorithm, see:

$\alpha^X \bmod \rho$

and

$\alpha^Y \bmod \rho$.

The Object Identifiers needed to register the key exchange method are shown in the following figure:

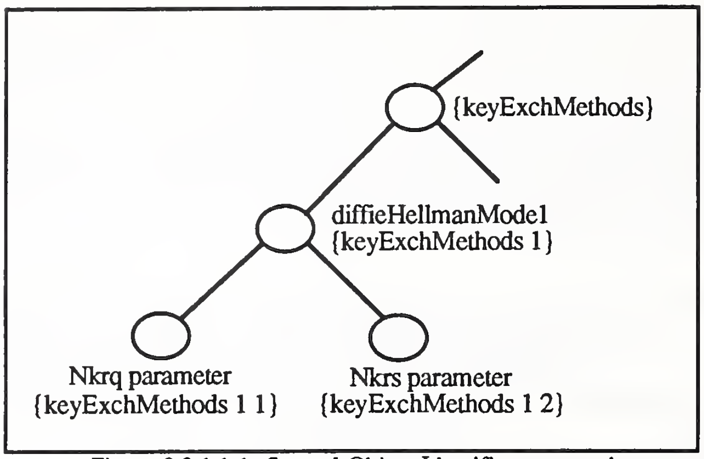


Figure 9.3.1.1:1 Several Object Identifiers are used

9.3.1.2 Approach two

Of course, the alternative is to use only one Object Identifier to register an algorithm. The macro used for this purpose is KEY-EXCH-ALG and is defined in the ASN.1 module KeyExchangeMethods. The definition of the macro is:

```
KEY-EXCHG-ALG MACRO ::=
BEGIN
   TYPE NOTATION ::=    ParameterData NkrqData NkrsData SsrqData SsrsData
                        RkrqData RkrsData RkdrqData RkdrsData
   VALUE NOTATION ::= VALUE(value OBJECT IDENTIFIER)
   ParameterData ::= empty | "PARAMETER" type
   NkrqData     ::=  "NkrqData" type
   NkrsData     ::=  "NkrsData" type
   SsrqData     ::=  empty | "SsrqData"       type
   SsrsData     ::=  empty | "SsrsData"       type
   RkrqData     ::=  empty | "RkrqData"       type
   RkrsData     ::=  empty | "RkrsData"       type
   RkdrqData    ::=  empty | "RkdrqData"      type
   RkdrsData    ::=  empty | "RkdrsData"      type
END
```

The purpose of the macro is to define the algorithm dependent data to fill the place-holders of the Nkrq, Nkrs, Ssrq, Ssrs, Rkrq, Rkrs, Rkdrq and Rkdrs PDUs. The empty declarations allow for the possibility of not associating any data with a specific place-holder. For example, neither [SDNS 90-4262] nor [GRAFF90] need to have algorithm dependent data in the Ssrq and Ssrs PDUs. However, the algorithm explained in Annex C of NLSP [ISO 11577] needs to have algorithm dependent data in at least the Ssrq PDU. When KEY-EXCHG-ALG is used, the key exchange algorithm presented in [GRAFF90] is registered in the following way:

```
diffieHellmanModel1 KEY-EXCHG-ALG
   PARAMETER SEQUENCE {
                           alpha INTEGER,
                           rho INTEGER}
   NkrqData SEQUENCE {
                           certPath CertificationPath,
                           unidirectKeys BOOLEAN OPTIONAL,
                           SIGNED EXPONENTIATION A P RandomA}
   NkrsData SEQUENCE {
                           certPath CertificationPath,
                           unidirectKeys BOOLEAN OPTIONAL,
                           SIGNED EXPONENTIATION A P RandomB}
::= {keyExchMethod 1}
```

The Object Identifier needed to register the key exchange method is illustrated by the following figure:

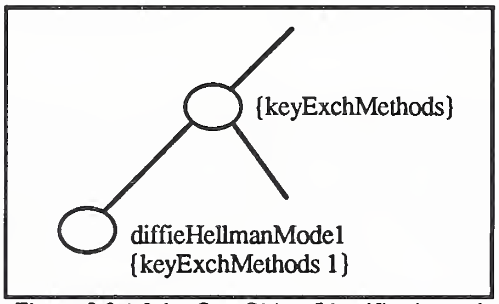
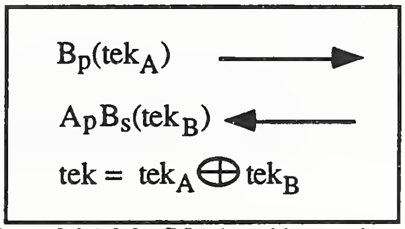


Figure 9.3.1.2:1 One Object Identifier is used

It is necessary to define which portions of the created bit string should be used as initialization vector, confidentiality key and integrity key. Such definitions are included in textual descriptions when a security object is registered.

So far, just a Diffie-Hellman based algorithm has been registered. It could be interesting to see whether other algorithms can be registered. The other algorithm chosen to register is RSA based, where the initiating key manager sends a string encrypted with the peers public key. The responding key manager returns a string encrypted with both the local private key and the initiator´s public key. The symmetric key material is derived by an exclusive-or operation on the two strings.


$B_p(tek_A) \longrightarrow$
$A_pB_s(tek_B) \longleftarrow$
$tek = tek_A \oplus tek_B$
Figure 9.3.1.2:2 RSA based key exchange

By exchanging the Ssrq and Ssrs PDUs encrypted with the key derived from the symmetric key material, the two parties can be convinced that they share the same secret key. This algorithm is registered in the following way:

```
rsaMode1 KEY-EXCHG-ALG
   PARAMETER TekSize
   NkrqData SEQUENCE {
                     certPath CertificationPath,
                     unidirectKeys BOOLEAN OPTIONAL,
                     ENCRYPTED RandomA}
   NkrsData SEQUENCE {
                     certPath CertificationPath,
                     unidirectKeys BOOLEAN OPTIONAL,
                     ENCRYPTED ENCRYPTED RandomB}
::= {keyExchMethod 2}

TekSize ::= INTEGER
```

It is assumed that the public key of B can be obtained from a directory service. An alternative is to send tek_A in the clear.

Because of ASN.1 a problem is encountered. For example, in order to describe the semantics of the RSA based key exchange, the encrypted strings are represented as ENCRYPTED RandomA and ENCRYPTED ENCRYPTED RandomB respectively. Both RandomA and RandomB are of ASN.1 data type INTEGER. When implementing this type of key exchange, these integers are never exposed outside the cryptofacility in clear text form. What is exposed outside the cryptofacility is their representation in encrypted form, that is, ENCRYPTED RandomA and ENCRYPTED ENCRYPTED RandomB. However, the cryptofacility must then perform the encryption on the transfer syntax representation of an ASN.1 INTEGER. That restriction requires the cryptofacility to add the type identifier and length information. In order to avoid this, the encrypted version of the numbers could be represented as a BIT STRING or an OCTET STRING. For example, a random number, tek_A, can be generated and encrypted. When the number is sent to the peer system, the ASN.1 identifier and length information can be added. However, the ASN.1 descriptions then will become less descriptive, see the following declaration:

```
   NkrqData SEQUENCE {
                     certPath CertificationPath,
                     unidirectKeys BOOLEAN OPTIONAL,
                     -- encrypted random number
                     OCTET STRING}
```

9.3.2 Registration of rekeying methods

The approach taken in the KMSA-KCA and KCA-KCA protocol specifications is to let rekeying take place between two peer entities. Other approaches were discussed in Section 5.6.1.2.

The protocol has been defined under the assumption that rekeying is initiated by a subordinate contacting its immediate superior.

As mentioned in 5.6.1.1 there are two different forms of rekeying, interactive and staged. The staged form of rekeying discussed in this section has similarities with [MATY86].

The relationship between the KMSA and its immediate superior KCA, or between a KCA and its immediate superior KCA is important. Can a KCA trust that a KMSA behaves in a proper manner? If we have a time period T_1-T_2 and a KMSA has to initiate the rekeying during this time period, can the KCA be sure that the KMSA does not make several attempts in order to get several new certificates? Must the KCA check the identity of the KMSA by manual means before issuing a new certificate? At least the two problems mentioned can be circumvented if there are two phases. A first phase in which a KMSA sends the public key for certification, and a second phase in which the certificate is collected. The Rkrq and Rkrs PDUs are used for the first phase and the Rkdrq and Rkdrs PDUs for the second. If some time periods need to be included, for example, delivery of the public key between T_1-T_2, and collection of the new certificate between T_2-T_3, they can be included in the ASN.1 declaration of the rekey method. If there is a need for it, additional data can also be included.

Again there are two different ways to register a method; one or several Object Identifiers may be used. The rekey method registered below is based on the principle that the user creates its private and public key pair, and sends the public component to the KCA for certification. The KCA returns a certificate when contacted at a later point in time.

The following ASN.1 declaration is used to register the rekey method:

```
rekWhenUserCreatesKey ALGORITHM
::= {rekeyingMethods 1}
```

The algorithm dependent data conveyed in the Rkrq PDU is the public key to be certified.

```
rekMeth1RkrqData ALGORITHM-ARGUMENT-REGISTRATION
   ARGUMENT SubjectPublicKeyInfo
::= {rekeyingMethods 1 1}
```

The algorithm dependent data conveyed in the Rkrs PDU is a transaction number.

```
rekMeth1RkrsData ALGORITHM-ARGUMENT-REGISTRATION
   ARGUMENT Transaction
::= {rekeyingMethods 1 2}
```

The algorithm dependent data conveyed in the Rkdrq PDU is the previously mentioned transaction number.

```
rekMeth1RkdrqData ALGORITHM-ARGUMENT-REGISTRATION
   ARGUMENT Transaction
::= {rekeyingMethods 1 3}

Transaction ::= INTEGER
```

The algorithm dependent data conveyed in the Rkdrs PDU is the certified public key. [ISO 9594-8] makes a distinction between a user certificate and a CA certificate. A certificate delivered to a KMSA is a user certificate, while a certificate delivered to a KCA is a CA certificate.

```
rekMeth1RkdrsData ALGORITHM-ARGUMENT-REGISTRATION
   ARGUMENT CHOICE {userCertificate [0] Certificate,
                    caCertificate [1] Certificate}
::= {rekeyingMethods 1 4}
```

The registration is illustrated below:

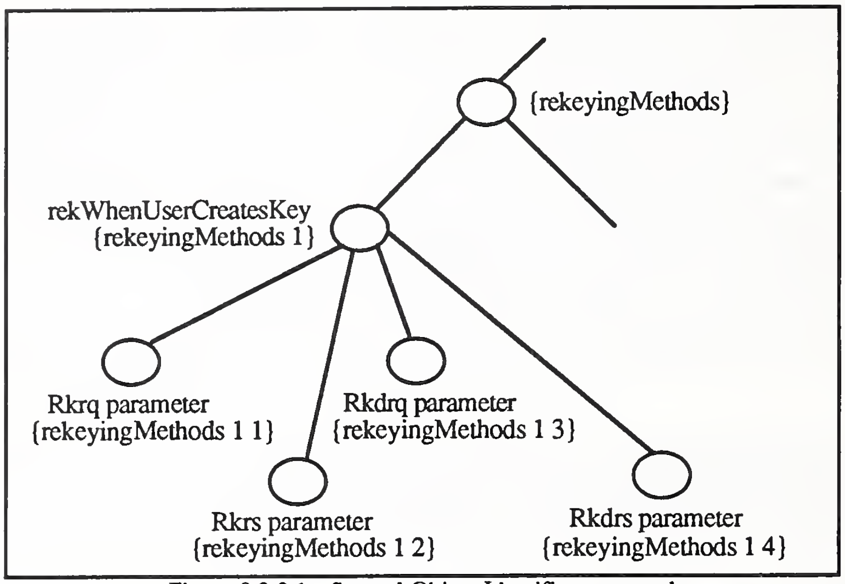


Figure 9.3.2:1 Several Object Identifiers are used

The following macro is used when the rekey method is registered with a single Object Identifier:

```
REKEY-REG MACRO ::=
BEGIN
   TYPE NOTATION ::=       ParameterData RkrqData RkrsData
                           RkdrqData RkdrsData
   VALUE NOTATION          ::= VALUE(value OBJECT IDENTIFIER)
   ParameterData      ::= empty | "PARAMETER" type
   RkrqData           ::= "RkrqData" type
   RkrsData           ::= "RkrsData" type
   RkdrqData          ::= empty | "RkdrqData" type
   RkdrsData          ::= empty | "RkdrsData" type
END
```

The Rkdrq and Rkdrs PDUs are only used for a staged form of rekeying. If the rekeying is interactive these PDUs are not used, hence the empty declarations. The rekey method is registered in the same way as already described in section 9.3.1. The value of the Object Identifier when rekWhenUserCreatesKey is registered by using macro REKEY-REG is *{rekeyingMethods 1}*.

If the [RFC 1114] scheme is followed, how should the new public key of the topmost KCA be delivered to a KMSA or to a KCA lower in hierarchy? This cannot be fully described by the model. The public key or certificate of the topmost KCA must be delivered to the KMSA under integrity protection. The Rkrq and Rkrs PDUs could be used to deliver the topmost public key. These PDUs are sent protected by the Ekpdu. However, can the delivery of a new public key be technically regarded as rekeying? The following declaration makes it possible for a KMSA to obtain the public key of the topmost KCA (CA):

```
getPublicKey REKEY-REG
   RkrqData NULL
   RkrsData SubjectPublicKeyInfo
::= {rekeyingMethods 2}
```

The declaration is a bit odd since the Object Identifier is used only to inform the receiver of the Rkrq PDU that the KMSA wants to obtain the public key, but the data associated with the Object Identifier, NULL, has no meaning.

9.4 PROTOCOL OVERVIEW

Figure 9.4:1 shows an example of a protocol overview. The state diagram shows the PDU mapping for the KMSA-KMSA protocol when a new security association is established. The KMSA-KCA and KCA-KCA protocol have a similar functionality.

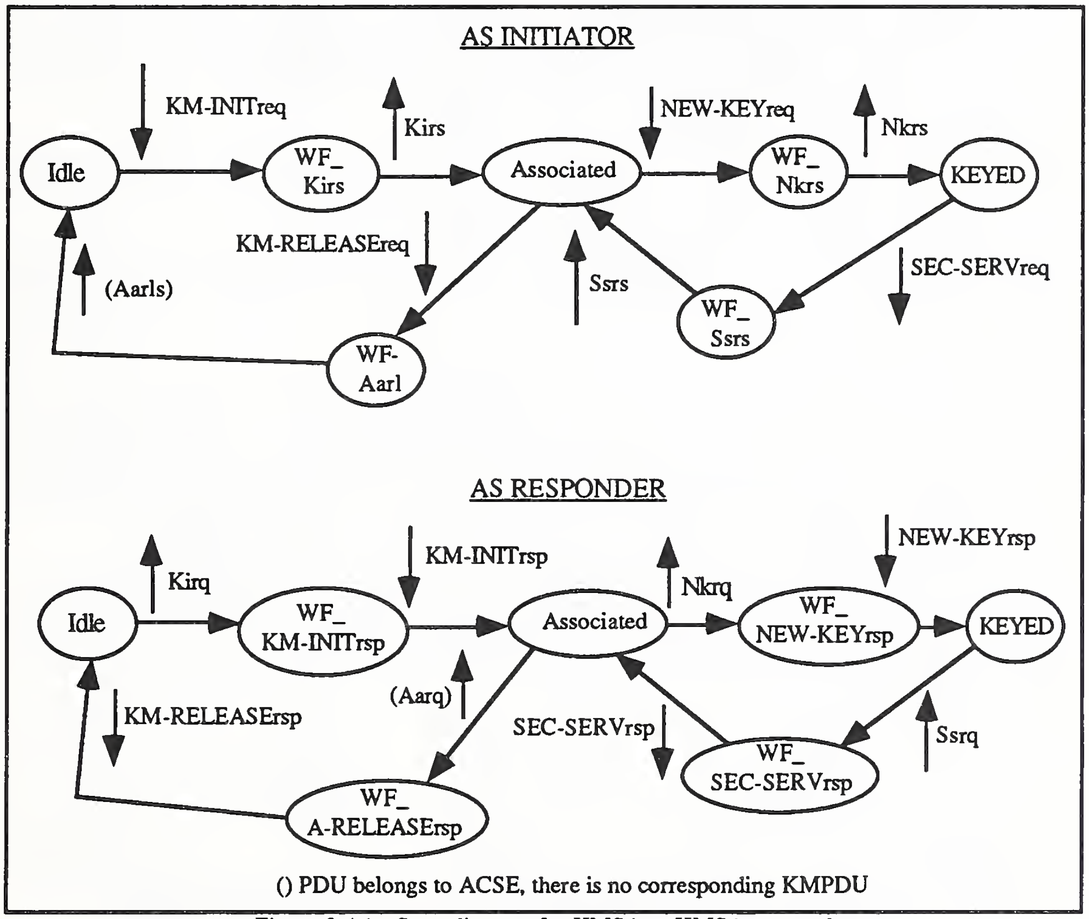


Figure 9.4:1 State diagram for KMSA to KMSA protocol

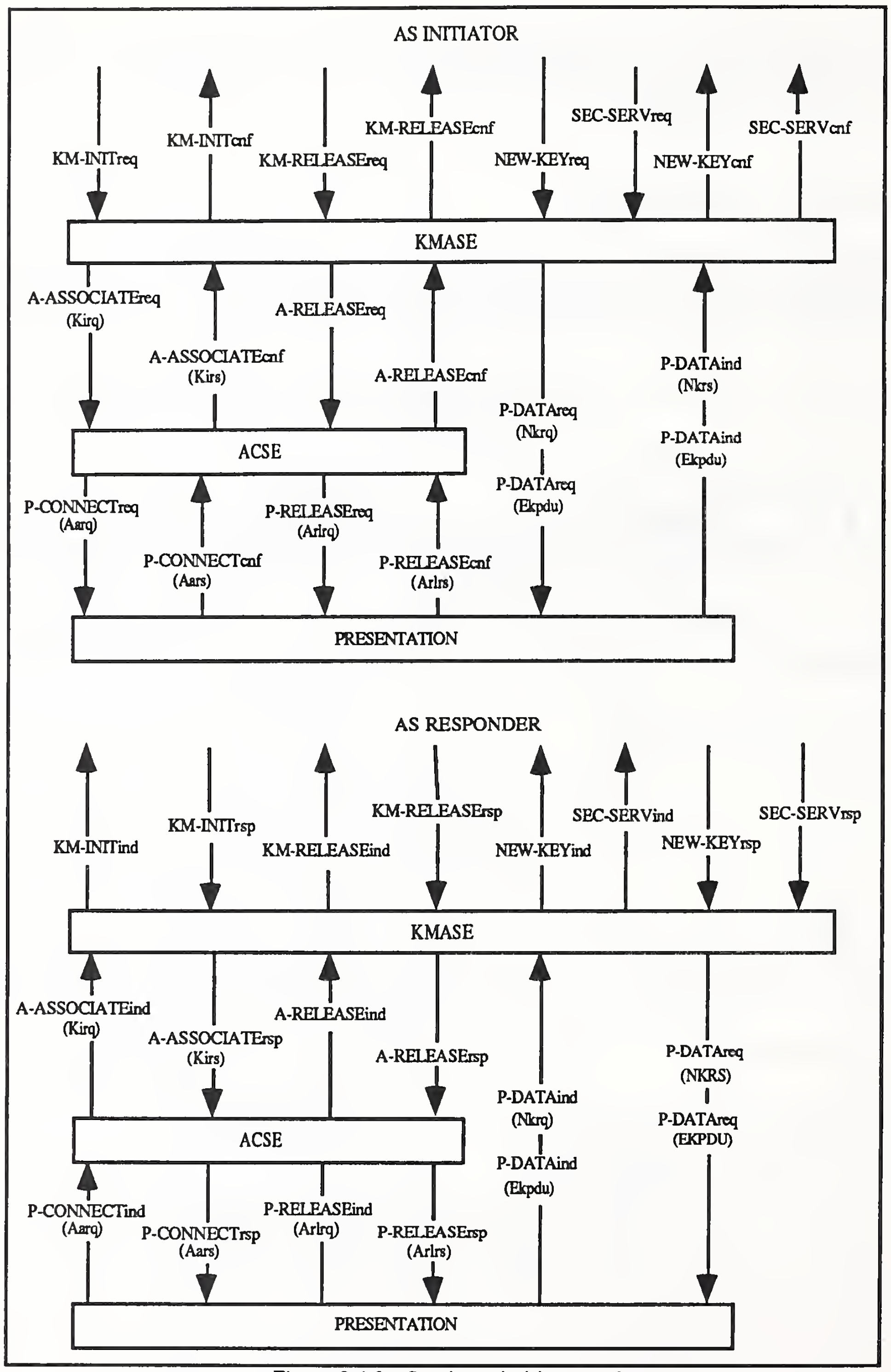


Figure 9.4:2 Service primitives used

The OSI aspects of an application process transpires in the application entity, see the Application Layers Structure document [ISO 9545] for more details. The key management service primitives can be seen as issued from the part of the key management application process that resides outside the application entity.

The initiating key management application process initiates a key exchange by issuing the service primitive KM-INITreq, which is sent to the Key Management ASE (KMASE). As a response, the peer key manager issues the KM-INITrsp service primitive. The KM-INIT service primitive is primarily used to open an application association, but it could also be used to perform algorithm negotiations. The Kirq PDU is derived from the parameters included in the KM-INITreq and the Kirs PDU is derived from the parameters included the KM-INITrsp. The KMASE maps the two PDUs onto the A-ASSOCIATE service primitive of ACSE. ACSE in turn uses the presentation layer services by mapping its Aarq and Aars PDUs onto the P-CONNECT service primitive. After an application association is established, the KMASE uses the presentation layer services directly. ACSE is used again when the application association has to be terminated. After the establishment phase, symmetric key material is created by using the NEW-KEY service primitive. The Ssrq and Ssrs PDUs are derived from the parameters included in the SECURITY SERVICE service primitive. Those PDUs are mapped to the Ekpdu which is confidentiality and integrity protected by the symmetric key material previously established. The last step is to terminate the application association by using the KM-RELEASE service primitive.

9.5 INTEGRITY PROCEDURE

The macro SEALED is used to describe integrity processing of key management protocol PDUs. The definition of the macro is:

```
SEALED MACRO      ::=
BEGIN
TYPE NOTATION     ::= type(ToBeProtected)
VALUE NOTATION           ::= value( VALUE
  SEQUENCE {
        SEQUENCE {
              ToBeProtected,
              padding OCTET STRING OPTIONAL},
        icv OCTET STRING
     -- where the OCTET STRING is icv size long and obtained by applying
     -- an integrity algorithm on SEQUENCE {ToBeProtected, padding}
        })
END -- of SEALED
```

The transfer syntax representation of:

```
  SEQUENCE {
        SEQUENCE {
              ToBeProtected,
              padding OCTET STRING OPTIONAL},
        icv OCTET STRING}
```

has to fit the block size of the encryption algorithm used. The *padding* field is used to adjust the size. If the algorithm is not block oriented *padding* can be omitted. The impression of the SEALED macro may be that it is too long and unnecessarily complicated. The corresponding definition in [SDNS 90-4262] is:

```
PlainTextKpdu ::= SEQUENCE {
   plaintext CHOICE {
            [0]   IMPLICIT    Ssrq,
            [1]   IMPLICIT    Ssrs,
            [2]   IMPLICIT    Estat,
            [3]   IMPLICIT    Kurq,
            [4]   IMPLICIT    Kurs,
            [5]   IMPLICIT    Rkrq,
            [6]   IMPLICIT    Rkrs,
            [7]   IMPLICIT    Srkrq,
            [8]   IMPLICIT    Srkrs,
            [9]   IMPLICIT    Srdrq,
            [10]  IMPLICIT    Srdrs,
            [11]  IMPLICIT    Crrq,
            [12]  IMPLICIT    Crrs},
   padding OCTET STRING OPTIONAL,
   icvIntegrityCheckValue}
```

However, it is felt that the integrity check value should be computed on a data structure. That is, something that can be regarded to be a unit, for instance, something encapsulated by a SEQUENCE declaration. That is not the case in the above declaration.

An example of an existing integrity algorithm is [ANSI X9.9].

9.6 ENCRYPTION PROCEDURE

The macro ENCRYPTED is used to describe the encryption procedure. The definition of the macro is:

```
ENCRYPTED MACRO        ::=
BEGIN
   TYPE NOTATION       ::= type(ToBeEnciphered)
   VALUE NOTATION      ::= value (VALUE BIT STRING)
END -- of ENCRYPTED
```

The data to be encrypted appears in *ToBeEnciphered*. The result of the encryption is an ASN.1 BIT STRING. The same macro is defined in [ISO 9594-8]. It has been redefined in order to have the macros describing confidentiality and integrity in the same ASN.1 module.

9.7 KEY EXCHANGE

The Nkrq and Nkrs PDUs convey the algorithm dependent data in the field *algArgument*. The ASN.1 declarations of the PDUs are:

```
Nkrq ::= SEQUENCE {
   algArgId        [0]  IMPLICIT OBJECT IDENTIFIER OPTIONAL,
   algArgument     [1]  IMPLICIT ANY DEFINED BY algArgId}

Nkrs ::= SEQUENCE {
   algArgId        [0]  IMPLICIT OBJECT IDENTIFIER OPTIONAL,
   algArgument     [1]  IMPLICIT ANY DEFINED BY algArgId}
```

The *algArgId* field is optional because there may be prior agreements about the algorithm to use. Chapter 9.3.1 proposes two different solutions of how to register the algorithm described in [GRAFF90]. Using the ALGORITHM-ARGUMENT-REGISTRATION macro to register the arguments, the field *algArgId* in the Nkrq carries Object Identifier value *{keyExchMethod 1 1}*. The corresponding field in the Nkrs carries *{keyExchMethod 1 2}*. *{keyExchMethod 1}* is the value notation for the Object Identifier used to register diffieHellmanMode1 as a key exchange method. When diffieHellmanMode1 is registered by the KEY-EXCH-ALG macro, the algArgId fields may carry *{keyExchMethod 1}*.

The field *algArgument* in the Nkrq when diffieHellmanMod1 is chosen, carries the following data:

```
SEQUENCE {
   certPath CertificationPath,
   unidirectKeys BOOLEAN OPTIONAL,
   SIGNED EXPONENTIATION A P RandomA}
```

The field *algArgument* in the Nkrs carries the following data:

```
SEQUENCE {
   certPath CertificationPath,
   unidirectKeys BOOLEAN OPTIONAL,
   SIGNED EXPONENTIATION A P RandomB}
```

If the [RFC 1114] scheme is followed, the two communicating KMSAs shares a common point of trust. The common point of trust is determined from the distinguished name of the KMSAs, herein called KMSA A and KMSA B. The distinguished name of a KMSA appears in the certificate of the KMSA. It is the lowest certificate in hierarchy of the certificates included in the certification path provided by the KMSA. At the common point of trust, both KMSA A and KMSA B are in possession of the public key of the associated KCA. By being in possession of that public key the certification path can be verified. From the certificate of the remote KMSA is extracted the public key. This public key is needed to verify the digital signature produced by the remote KMSA. KMSA A has to verify:

SIGNED EXPONENTIATION A P RandomB

After the signature is verified, KMSA A derives the key string by taking the value sent by KMSA B, *EXPONENTIATION A P RandomB* ($\alpha^Y \bmod \rho$), and performs the operation:

$(\alpha^Y \bmod \rho)^X \bmod \rho = \alpha^{XY} \bmod \rho$.

KMSA B performs the corresponding operation.

The authentication information is carried by the Nkrq and Nkrs PDUs; there is no need to include authentication information in the Ssrq and Ssrs PDUs. However, certain algorithms, for example, the one described in NLSP [ISO 11577], need to include authentication information in at least the Ssrq. The definitions of the Ssrq and Ssrs PDUs are:

```
Ssrq ::= SET {
   algDepData [0] IMPLICIT SET {
            algArgId          [0]  IMPLICIT OBJECT IDENTIFIER OPTIONAL,
            algArgument       [1]  IMPLICIT  ANY DEFINED BY algArgId} OPTIONAL,
   said [1] Said,
   spParameters CHOICE {
              [2]  IMPLICIT TLSPParams,
              [3]  IMPLICIT NLSPParams,
              [4]  IMPLICIT SDEParams}}

Ssrs ::= Ssrq
```

The declaration:

```
SET {
   algArgId          [0]  IMPLICIT OBJECT IDENTIFIER OPTIONAL,
   algArgument       [1]  IMPLICIT ANY DEFINED BY algArgId} OPTIONAL
```

is made optional because it needs only to be present when algorithm dependent data is required.

In an asymmetric form of key management the KMSA-KCA and KCA-KCA protocols are used to establish the new private key and certificate, alternatively just the certificate, of a KMSA and KCA respectively. In order to establish a confidentiality and integrity protected channel, the Nkrq and Nkrs PDUs are used to create the needed key material. Since the KMSA-KCA and KCA-KCA protocols are not intended to create keys and attributes for security protocols, they do not include the Ssrq and Ssrs PDU definitions. Instead, they contain the Rkrq, Rkrs, Rkdrq and Rkdrs PDUs.

9.8 KMSA TO KMSA PROTOCOL

The KMSA-KMSA protocol specifies the interaction between two KMSAs. The mission of the KMSA-KMSA protocol is to establish the keys and associated attributes needed by the two communicating security protocols, or to update the security association.

9.8.1 Algorithm choice

There are three different possibilities for two KMSAs to establish the algorithms to use:

- The KMSAs have a priori agreements;
- The KMSAs have prior agreements but need to initialize cryptographic variables;
- The KMSAs negotiate about the algorithms to use.

The protocol uses the Kirq and Kirs PDUs to establish the algorithms.

```
Kirq ::= SEQUENCE {
   protocolVersion ProtocolVersion,
   secServGrpProp [0] IMPLICIT SecServGrpProp OPTIONAL,
   keyEchGrpProp CHOICE {
     [1]  IMPLICIT AsymmKeyExchGrpProp,
     [2]  IMPLICIT SymmKeyExchGrpProp } OPTIONAL }
```

```
Kirs ::= SEQUENCE {
    stateResult StateResult,
    protocolVersion ProtocolVersion,
    secServGrpAcc [0] IMPLICIT SecServGrpAcc OPTIONAL
    keyEchGrpAcc CHOICE {
        [1]  IMPLICIT AsymmKeyExchGrpAcc,
        [2]  IMPLICIT SymmKeyExchGrpAcc } OPTIONAL}

SecServGrpProp ::= SEQUENCE {
    confAlgs SEQUENCE OF AlgorithmIdentifier,
    icvAlgs SEQUENCE OF AlgorithmIdentifier}

SecServGrpAcc ::= SEQUENCE {
    confAlg AlgorithmIdentifier,
    icvAlg AlgorithmIdentifier}

AsymmKeyExchGrpProp ::= SEQUENCE {
    keyExchMethods SEQUENCE OF AlgorithmIdentifier
-- possibly more declarations have to be put in}

AsymmKeyExchGrpAcc ::= SEQUENCE {
    keyExchMethod AlgorithmIdentifier
-- possibly more declarations have to be put in}
```

9.8.1.1 A priori agreements

Both *secServGrpProp* and *keyEchGrpProp* are optional because there can be prior agreements between the KMSAs about the algorithms to use. The KMSAs may therefore have no need to send the information contained in secServGrpProp and keyEchGrpProp. Before the initiating key manager performs a key exchange, it looks up in the SMIB whether there are prior agreements made with the peer system.

Problems arise when a KMSA makes a priori agreements with several other KMSAs and the agreements include different types of algorithms. For example, KMSA A has made an agreement to use key exchange method diffieHellmanMode1 with KMSA B and the key exchange method described in NLSP with KMSA C. Since algorithm identifiers are not exchanged when the application association is opened, how can KMSA A when acting as the responding party know the syntax of the algorithm dependent data conveyed in the Nkrq PDU? This is because KMSA A does not yet know the identity of the initiating party when the Nkrq is received. One solution is to do a table look-up based on the remote party´s presentation address; thereby finding out the key exchange method going to be used. This look-up has to be performed at the creation of the application association, thus, before the Nkrq arrives. The presentation address uniquely distinguishes the remote key management entity. A second possibility is to include the distinguished name of a key management entity, alternatively a domain identifier, in the Kirq and Kirs PDUs. A third possibility is to include the Object Identifier value in the Nkrq PDU. This Object Identifier is declared as an optional parameter. The intention is that if the algorithms are preestablished, it has to be possible locally to determine and apply the Object Identifier once the association is established with the peer key management entity. The way in which the translation from transfer to abstract syntax is done is important. If the translation is done in the presentation layer, when an Nkrq PDU arrives encapsulated in a presentation PDU, the ASN.1 parser must know the value of the Object Identifier, otherwise it cannot make the translation from transfer to abstract syntax.

9.8.1.2 Need to initialize cryptographic variables

The two communicating KMSAs may know the algorithms to use but they need to initialize some cryptographic variables. For example, the values a and r in Diffie-Hellman can be preestablished or they can be unique for each session. In the previous section it was assumed that the cryptographic variables were preestablished. Discussions Of Symmetrical (Secret Key) And Asymmetrical (Public Key) Cryptography [BARK91] suggests to have r fixed because it needs to be a strong prime number. The motive is that creating a strong prime number is too time consuming. It should therefore be created in advance and not during the session. Annex C of NLSP [ISO 11577] suggests to have a and r included in the exchange. This would make it possible to make them unique for each session. If the security domain chooses the latter solution, the following described protocol definitions make it possible to make them unique on a per session basis. The PARAMETER field of the KEY-EXCH and ALGORITHM macros are used to register the cryptographic variables. When diffieHellmanMode1 was registered by using the KEY-EXCH macro, the registration was done in the following way:

```
diffieHellmanMode1 KEY-EXCHG-ALG
   PARAMETER SEQUENCE {
                           alpha INTEGER,
                           rho INTEGER}
   NkrqData SEQUENCE {
                           certPath CertificationPath,
                           unidirectKeys BOOLEAN OPTIONAL,
                           SIGNED EXPONENTIATION A P RandomA}
   NkrsData SEQUENCE {
                           certPath CertificationPath,
                           unidirectKeys BOOLEAN OPTIONAL,
                           SIGNED EXPONENTIATION A P RandomB}

::= {keyExchMethod 1}
```

The initiating key management entity can include a and r in *AlgorithmIdentifier*. AlgorithmIdentifier is included in *keyExchMethods* which is a subfield of the ASN.1 data type *AsymmKeyExchGrpProp*. AsymmKeyExchGrpProp is part of the Kirq PDU. However, the initiating entity must then also include the Object Identifier value of *algorithm* which is included in AlgorithmIdentifier. The possibility exists to have "algorithm" as an optional field. It is not declared optional because the AlgorithmIdentifier as defined in [ISO 9594-8] is used. Since the algorithms are preestablished only one algorithm identifier needs to be included in *keyExchMethods*.

```
AsymmKeyExchGrpProp ::= SEQUENCE {
   keyExchMethods SEQUENCE OF AlgorithmIdentifier,
   -- possibly more declarations have to be put in}
```

The ASN.1 data type AlgorithmIdentifier has the following definition:

```
AlgorithmIdentifier ::= SEQUENCE {
                              algorithm OBJECT IDENTIFIER,
                              parameter ANY DEFINED BY algorithm}
```

The *parameter* field will contain:

```
SEQUENCE {
   alpha INTEGER,
   rho INTEGER}.
```

Allowing a unique a and r for each session may lead to another problem. Can the receiver trust that a good value for r is chosen? If the values are preestablished, such a trust does exist.

9.8.1.3 Algorithm negotiation

If there are no agreements between the KMSAs, they must perform an algorithm negotiation. It is up to the local security policy to decide whether all algorithms available in the system can be revealed during the negotiation phase.

The ASN.1 variable *secServGrpProp* contains the confidentiality and integrity algorithms proposed by the initiating key management entity. The sequences are ranked in preferred order (most preferred is first in sequence).

```
secServGrpProp [0] IMPLICIT SecServGrpProp OPTIONAL

SecServGrpProp ::= SEQUENCE {
   confAlgs SEQUENCE OF AlgorithmIdentifier,
   icvAlgs SEQUENCE OF AlgorithmIdentifier}
```

The responding key manager selects one of the algorithms proposed:

```
secServGrpAcc [0] IMPLICIT SecServGrpAcc OPTIONAL

SecServGrpAcc ::= SEQUENCE {
   confAlg AlgorithmIdentifier,
   icvAlg AlgorithmIdentifier}
```

The ASN.1 variable *keyEchGrpProp* indicates whether an asymmetric or symmetric algorithm is employed. The choice is made in the following declaration:

```
keyEchGrpProp CHOICE {
   [1] IMPLICIT AsymmKeyExchGrpProp,
   [2] IMPLICIT SymmKeyExchGrpProp}

AsymmKeyExchGrpProp ::= SEQUENCE {
   keyExchMethods SEQUENCE OF AlgorithmIdentifier
-- possibly more declarations have to be put in}
```

No definitions are made for a symmetric form of key management, but the variable *SymmKeyExchGrpProp* is designated for the symmetric definitions. A question is whether special PDU types should be defined for the symmetric form of key management. The service negotiation phase (performed by exchanging the Ssrq and Ssrs PDUs) can still be the same for the two schemes. Should both schemes use the same PDUs for the key establishment phase? If so, the Nkrq and Nkrs PDUs are also used for symmetric purposes. This is possible because the protocol is algorithm independent. Since [ANSI X9.17] includes a lot of options; it is probably easier to proceed with the symmetric definitions by eliminating some of the seldom used X9.17 options. If the solution is to use generic PDU types, it is felt that the current names of the PDUs imply an asymmetric form of key management. The names should instead be made applicable for both an asymmetric and symmetric form of key management. On the other hand, if a separation is made, the protocol becomes more clear and easy to read. See also section 9.17.

The above ASN.1 definitions do not allow a mix of asymmetric and symmetric key management. The idea was that in an implementation either of the approaches is selected. However, often standards do allow a great deal of flexibility, it is up to the ones implementing a standard to choose between the different options offered. The following ASN.1 definitions allow a mix of asymmetric and symmetric key management.

```
keyEchGrpProp SEQUENCE OF CHOICE {
            [1]  IMPLICIT AsymmKeyExchGrpProp,
            [2]  IMPLICIT SymmKeyExchGrpProp}

AsymmKeyExchGrpProp ::= SEQUENCE {
   keyExchMethods AlgorithmIdentifier
-- possibly more declarations have to be put in}
```

The proposed scheme does not relate a proposed key exchange algorithm to a particular proposed encryption and integrity algorithm. If instead the algorithms should be regarded as a unit, the following definitions are more appropriate:

```
PropAlgPackage ::= SEQUENCE OF SEQUENCE {
                              keyExchAlg     AlgorithmIdentifier,
                              encrAlg        AlgorithmIdentifier,
                              integrAlg      AlgorithmIdentifier}
AccAlgPackage ::= SEQUENCE {
                              keyExchAlg     AlgorithmIdentifier,
                              encrAlg        AlgorithmIdentifier,
                              integrAlg      AlgorithmIdentifier}
```

In the case of Diffie-Hellman, how do the KMSAs know in which finite field[1] the computations are done? If the document type approach is chosen, such kind of information can be included in the document type. This further emphasizes the need of a textual description in the register. The KMSAs also need to agree on the size of r.

Certificate formats may also be negotiated. The protocol definitions are made under the assumption that certificates described as in [ISO 9594-8] are used. However, a security domain might want to include other type of information in a certificate, and therefore defines its own certificate format.

During the the two KMSAs negotiation, the situation can occur that they do not have algorithms in common. The *stateResult* parameter in the Kirs PDU is used to detect this situation. If the stateResult parameter indicates the failure value, the next state of the protocol machine is not reached.

[1] For more details about finite fields see Appendix G of Public Key Cryptography [67].

9.8.2 Update of traffic encryption keys

The Kurq and Kurs PDUs are used to perform a key update. The definition of the PDUs are:

```
Kurq ::= SEQUENCE {
   remSaid Said,
   ENCRYPTED SEALED newLocSaid Said}
Kurs ::= SEQUENCE {
   remSaid Said,
   ENCRYPTED SEALED newLocSaid Said}
```

The initiating entity conveys in the Kurq PDU the new value of the security association identifier. The Kurs PDU contains the new security association identifier of the responding entity.

If there is no prior knowledge about the key update method to use, new keys must be created by using the Nkrq and Nkrs PDUs. The alternative is to include the key update algorithms in a security register. There is also a need of a delete security association primitive. This to avoid the situation where the security protocol entity with the longer cryptoperiod encrypts PDUs with an encryption key that is no longer valid at the other side. The situation may occur when the two KMSAs belong to different security domains and do not have a common key update method. If algorithm identifiers are used for the KMSAs to agree on a key update method, the algorithm identifiers can be part of the Kirq and Kirs PDUs:

```
updateProp SEQUENCE OF AlgorithmIdentifier -- to be in Kirq
updateAcc AlgorithmIdentifier -- to be in Kirs
```

The PDU used to inform peer that a security association is deleted could have the following definition:

```
DelSaid ::= SEQUENCE {
                remSaid Said,
                ENCRYPTED SEALED LocalSaid}
```

LocalSaid would be of type Said.

9.8.3 Support of different kinds of security protocols

The ASN.1 declarations associated with a particular security protocol are defined in a separate ASN.1 module, see the figure below:

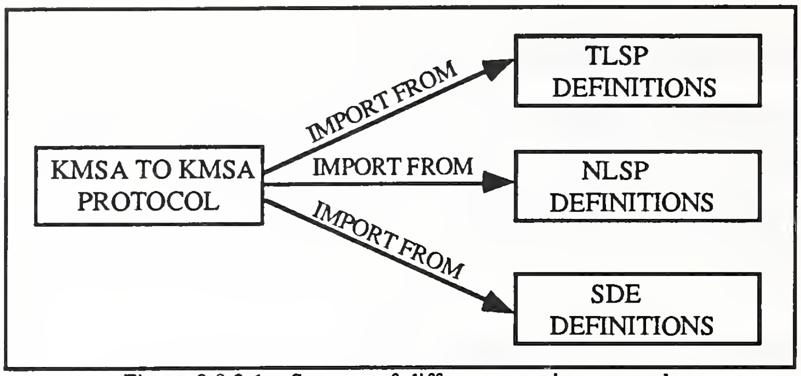


Figure 9.8.3:1 Support of different security protocols

In the Ssrq and Ssrs PDUs, the following CHOICE declaration makes it possible to select the desired security protocol:

```
spParameters CHOICE {
                    [2] IMPLICIT    TLSPParams,
                    [3] IMPLICIT    NLSPParams,
                    [4] IMPLICIT    SDEParams}
```

TLSPParams, NLSPParams and SDEParams are defined in separate ASN.1 modules and are imported by the KMSA-KMSA protocol. The above definition makes it possible for a single key manager to support several security protocols.

The Ssrq as defined in Annex B does not provide the possibility of referring to an already established security association. If such a possibility exists, there is no need to renegotiate about the attributes of a security protocol when a new security association is created with the characteristics of an existing one. This possibility is given in [SDNS 90-4262]. The following definition includes this capability:

```
spParameters CHOICE {oldSaid [1] Said,
                              newNeg [2] CHOICE {
                                     [3]  IMPLICIT    TLSPParams,
                                     [4]  IMPLICIT    NLSPParams,
                                     [5]  IMPLICIT    SDEParams}}
```

9.8.4 Security association identification

A security association identifier (SAID) is used by a security protocol to recognize the appropriate security association. Each key manager names the security association, that is, each security association has two SAIDs one assigned by each key manager. The Ssrq and Ssrs PDUs convey the assignment to the remote key manager.

```
Ssrq ::= SET {
   algDepData [0] SET {
         algArgId            [0]   IMPLICIT OBJECT IDENTIFIER OPTIONAL,
         algArgument         [1]   IMPLICIT ANY DEFINED BY algArgId} OPTIONAL,
   said [1] Said,
   spParameters CHOICE {
            [2]  IMPLICIT    TLSPParams,
            [3]  IMPLICIT    NLSPParams,
            [4]  IMPLICIT    SDEParams}}

Ssrs ::= Ssrq
```

9.8.5 Separation of confidentiality and integrity keys

Should the key manager provide the possibility to separate the encryption of key management PDUs from the encryption of security protocol PDUs? Different keys or even different algorithms may be used. One set to protect key management PDUs and the other to protect security protocol PDUs. The KMSA-KMSA protocol is specified with the assumption that the same keys (and thereby the same algorithms) are used to protect the key management Ssrq and Ssrs PDUs and security protocol PDUs.

9.9 KMSA TO KCA PROTOCOL

The KMSA-KCA protocol specifies the interactions between a KMSA and its immediate superior KCA. For asymmetric key management the result of the interactions is the new private key and certificate, alternatively just the certificate, of a KMSA. For symmetric management the result is notarized data key(s) possibly in combination with an automatically distributed Key Encryption Key.

The following description covers the asymmetric form of key management. No declarations are made for the symmetric form.

9.9.1 Algorithm choice

The Kirq and Kirs PDUs are used to perform the algorithm negotiation, the definitions of the PDUs are:

```
Kirq ::= SEQUENCE {
   protocolVersion ProtocolVersion,
   secServGrpProp [0] IMPLICIT SecServGrpProp OPTIONAL,
   keyExchGrpProp CHOICE {
                  [1]  IMPLICIT AsymmKeyExchGrpProp,
                  [2]  IMPLICIT SymmKeyExchGrpProp} OPTIONAL}

Kirs ::= SEQUENCE {
   stateResult StateResult,
   protocolVersion ProtocolVersion,
   secServcGrpAcc [0] IMPLICIT SecServGrpAcc,
   keyExchGrpAcc CHOICE {
                  [1]  IMPLICIT AsymmKeyExchGrpAcc,
                  [2]  IMPLICIT SymmKeyExchGrpAcc} OPTIONAL}
```

The declarations are almost identical to the ones of section 9.8.1. An ability to negotiate the rekeying method is supported:

```
AsymmKeyExchGrpProp ::= SEQUENCE {
   keyExchMethods          [0]  IMPLICIT SEQUENCE OF AlgorithmIdentifier,
   rekeyMethods            [1]  IMPLICIT SEQUENCE OF AlgorithmIdentifier}
                                                                OPTIONAL

AsymmKeyExchGrpAcc ::= SEQUENCE {
   keyExchMethod           [0]  IMPLICIT AlgorithmIdentifier,
   rekeyMethod             [1]  IMPLICIT AlgorithmIdentifier} OPTIONAL
```

9.9.2 Rekeying

The Nkrq and Nkrs PDUs are used as in the KMSA-KMSA protocol. The created keys confidentiality and integrity protect the Ekpdu:

```
Ekpdu ::= ENCRYPTED SEALED CHOICE {
                  [0] IMPLICIT     Rkrq,
                  [1] IMPLICIT     Rkrs,
                  [2] IMPLICIT     Rkdrq,
                  [3] IMPLICIT     Rkdrs}
```

The Rkrq, Rkrs, Rkdrq, and Rkdrs PDUs contain two kinds of algorithm dependent data. One is associated with the authentication, the other with the rekey method.

```
Rkrq ::= SET {
        auth        [0]  IMPLICIT SET{
          algArgId        [0]   IMPLICIT OBJECT IDENTIFIER OPTIONAL,
          authData        [1]   IMPLICIT ANY DEFINED BY algArgId} OPTIONAL,

          rek       [1]  IMPLICIT SET{
            algArgId         [0]   IMPLICIT OBJECT IDENTIFIER OPTIONAL,
            rekData          [1]   IMPLICIT ANY DEFINED BY algArgId}}
```

The Rkrs, Rkdrq and Rkdrs PDUs have identical definition.

When a KMSA initiates a rekey operation, the field *algArgId* of *rek* identifies the rekey method. If the macro REK-REG is used to register the rekey method a possible value for algArgId is *{rekeyingMethods 1}*. Then the rekData field of the Rkrq PDU contains:

SubjectPublicKeyInfo

SubjectPublicKeyInfo is defined in [ISO 9594-8], it is the public key to be certified.

The Rkrs PDU is sent as a response to the Rkrq PDU. The *rekData* field as an optional parameter contains:

Transaction (which is of type INTEGER)

When the requesting KMSA later sends the Rkdrq PDU to collect the certificate, the transaction integer is included in the PDU.

The *rekData* field of the Rkdrs PDU contains the certificate:

```
CHOICE {userCertificate        [0] Certificate,
           caCertificate        [1] Certificate}
```

For the KMSA-KCA protocol definition only the userCertificate alternative is valid. But by including both options, a distinction between the KMSA-KCA and KCA-KCA protocol is made. Both protocol can use the same registered method. The syntax of Certificate is defined in [ISO 9594-8].

9.10 KCA TO KCA PROTOCOL

The KCA-KCA protocol establishes the new private key and certificate, alternatively just the certificate, of a KCA. For symmetric key management the protocol transfers subscriber keys between two centers.

No definitions are included in Annex B concerning the KCA-KCA protocol. Probably the definitions will be very similar to the ones of the KMSA-KCA protocol.

9.11 SECURITY PROTOCOL DEFINITIONS

Three separate ASN.1 modules may contain the security protocol definitions:

- An ASN.1 module designated for the IEEE SDE protocol.
- An ASN.1 module designated for NSLP.
- An ASN.1 module designated for TLSP.

The security protocol definitions included in Annex B concern only TLSP. The SDE and NLSP ASN.1 modules could easily be added.

The ASN.1 definitions of the KMSA-KMSA protocol associated with a particular security protocol are included in the Ssrq and Ssrs PDUs. These PDUs are confidentiality and integrity protected:

```
Ekpdu ::= ENCRYPTED SEALED CHOICE {
   [0]  IMPLICIT    Ksrq,
   [1]  IMPLICIT    Ksrs,
   [2]  IMPLICIT    Ssrq,
   [3]  IMPLICIT    Ssrs}
```

The Ksrq and Ksrs PDUs were originally included as an alternative to make a six-way handshake procedure possible. It is now felt that these PDUs are unnecessary.

9.11.1 TLSP definitions

The ASN.1 definitions needed to support TLSP are:

```
TLSPParams ::= SET {
        confid       [0]  IMPLICIT BOOLEAN,
        integr       [1]  IMPLICIT BOOLEAN,
        secLabelDef        [2]  CHOICE {
                             propLabels       [0]  IMPLICIT PropLabels,
                             accLabel         [1]  IMPLICIT AccLabel} OPTIONAL,
        kindOfKey INTEGER{(0) perNSAP, (1) perTConnection}}

END

PropLabels ::= SET OF {
              secLevelSet              [0]  IMPLICIT SecLevelsIdentifier,
              secLabelType             [1]  IMPLICIT OBJECT IDENTIFIER}

AccLabel ::= SET {
              secLevelSet              [0]  IMPLICIT SecLevelsIdentifier,
              secLabelType             [1]  IMPLICIT OBJECT IDENTIFIER}
```

confid indicates whether a confidentiality service has to be included in the security association.

integr indicates whether an integrity service has to be included in the security association.

secLabelDef associates a security level set with a security label format. It is not clear how to register security transport layer label formats because the registrations are made using ASN.1.

kindOfKey tells which flavor of TLSP that has to be used, on a per connection or on a per NSAP basis.

SecLevelsIdentifier has the same ASN.1 definition as AlgorithmIdentifier.

9.12 ERROR HANDLING

Some suggestions are made in the protocols definitions concerning error handling. In each state of the protocol machines, except in the IDLE states, there is a possibility for the key management application process to issue an ABORT service primitive. The service primitive is issued when an error occurs. It is up to the local security policy to decide which diagnostic parameters are allowed. Diagnostic parameters could provide useful information to an adversary. The protocol definitions contain some suggestions about the diagnostic parameters.

9.13 BROADCAST AND GROUP KEYS

No attempt is made to solve the problem of creating keys shared between more than two entities. TLSP does not need this functionality.

9.14 FINDING THE REMOTE ADDRESS DYNAMICALLY

No attempt is made to solve the problem of dynamically finding the presentation address of the remote key manager. TLSP requires key management to be collocated at an end system, therefore TLSP does not need this functionality.

9.15 TIMERS

No declarations concerning timers are made. First, it should be investigated how timers used elsewhere in the protocol stack, such as transport, can be used. Second, it is unresolved whether this issue should be reflected in a key management standard, or it should be regarded as an implementation issue.

9.16 POSSIBILITY TO ENHANCE SECURITY

Should the remote key management entity be able to enhance the security? This could be done by including more options in the Ssrs than in the Ssrq. In this paper it is assumed so. However, according to the SQOS approach, see section 5.13, it is not possible to enhance security at connection establishment. [SDNS 90-4262] contains the following definitions:

```
Ssrq ::= SET {
   crl-ver        Crl-Ver,
                  NewOrOldOptions,
   add-info       AddInfo OPTIONAL }

Ssrs ::= SET {
   crl-ver               Crl-Ver,
   selected-options           [0]  AttributeSet,
   add-info                   [1]  IMPLICIT AddInfo OPTIONAL }

NewOrOldOptions       CHOICE {
   old-resp-kid                 KeyId,
   proposed-options             SEQUENCE OF AttributeSet }
```

The ASN.1 variable *proposed-options* indicates that the initiating key management entity can propose several alternatives. The responding entity selects one of the alternatives. Thus, the responding entity cannot enhance the security level.

9.17 GENERALITY OF THE KEY MANAGEMENT PROTOCOL

The declaration:

```
keyEchGrpAcc CHOICE {
   [1] IMPLICIT AsymmKeyExchGrpAcc,
   [2] IMPLICIT SymmKeyExchGrpAcc} OPTIONAL
```

shows where the choice between a symmetric and asymmetric form of key management is made. It is very simple from an ASN.1 point of view. Such a simple ASN.1 declaration can change the key management philosophy completely. Either, a symmetric form of key management based on a centralized third party, the Key Distribution Center, or, an asymmetric form of key management which is distributed, can be chosen. How general should the key management protocol be? By having a very restrictive protocol, where certain solutions are presumed, more is expressed about the underlying key management philosophy. In OSI security standardization, general solutions are preferred. It is up to the local security policy that is using the protocol to decide about the underlying philosophy.

The following illustrates the level of generality. X9.17, is the most restrictive, and Security Exchange ASE is the most general.

X9.17

ISO 8732

This protocol specification

Security Exchange ASE

[ANSI X9.17] specifies a symmetric form of key management for use with the DES.

ISO 8732 specifies a symmetric form of key management, but it does not mention the algorithm to use. It tries to achieve algorithm independence by separating the algorithm and the protocol used to distribute the keys.

The protocol specifications defined in this chapter outline how both the symmetric and asymmetric forms of key management can be incorporated into the same protocol. The protocol is solely intended for key management.

The Security Exchange ASE (SE ASE) [ISO/SC21 6096] [ISO/SC21 6097] [ISO/SC21 6098], where almost everything can be done from a security point of view, is the most generic one. [ISO/SC21 6096] states that the SE ASE could provide: authentication services, access control services, confidentiality services, non-repudiation services and *key management services*.

Is a high degree of generality always preferable? For authentication, confidentiality, and integrity a generic ASE used by different kinds of application protocols is probably preferable. For example, the FTAM standard would not need to include its own strong authentication procedure; instead, it could rely on a generic authentication service. However, key management is so complex that it can be a drawback to rely on a generic security service ASE. A key management protocol describes the things that are unique to key management, such as:

- It includes a state machine especially designated for the key management protocol.
- It includes in one place all the non algorithm specific declarations.
- The names of the PDUs reflects some key management aspect.

Intuitively, when key management is included in a generic ASE, the protocol readers understanding of key management diminishes.

10 COMPARISONS APPLICATION LAYER VERSUS SECURITY PROTOCOL LAYER

This chapter lists some pros and cons of having key management in a security protocol versus having key management in the application layer as an ASE.

10.1 APPLICATION LAYER ADVANTAGES

The key management protocol supports different types of security protocols, thereby the key management functionality needs to be at only one place in the OSI architecture.

The problems concerned with the underlying network are avoided. If the network is connectionless, it does not affect the application layer key manager.

ASN.1 data types provided by the application layer can be used to achieve algorithm independence. Objects in security registers will most likely be registered using ASN.1.

The use of ASN.1 algorithm identifiers allows the possibility to perform algorithm negotiations. This possibility is not included in NLSP.

SDNS keeps the security protocols (SP3 and SP4) simple and independent of a specific key management scheme. To include key exchanges in a lower layer security protocol, significantly increases the complexity of the security protocol.

10.2 SECURITY PROTOCOL ADVANTAGES

Less overhead is incurred by having key management as part of the security protocol. For example, if an application layer key manager creates keys and attributes for NLSP, the key manager must open a transport connection, session connection, presentation connection and application association. If the functionality resides in the network layer this overhead is avoided.

The problems concerned with the abstract versus transfer syntax representation of ASN.1 are also avoided. In layers below presentation the data is represented in the form of octets, which have no semantic meaning. That is, they are not regarded as an Octet String, Bit String, etc. There is no need of any translations to occur. The application layer data is represented in its abstract syntax form and needs to be translated.

If key management is included in the security protocol, it is probably easier to implement the security protocol and its associated key management functionality in form of a black box which can be trusted from a security point of view.

Including key management in the security protocol gives a short term solution. At the moment no internationally standardized layer seven key management protocol is available.

11 AREAS NOT COVERED

This chapter lists areas felt to be important but that have not been covered or fully covered in this paper.

11.1 SYMMETRIC FORM OF KEY MANAGEMENT

The protocol specifications included in Annex B contain only an asymmetric form of key management. The key management protocol would be more complete if both options are included. The protocol needs to be augmented with the symmetric part.

11.2 REKEY HANDLING

The rekeying definitions are included in the key management protocol to make the asymmetric part of the protocol complete. Much more work needs to be done to define a good rekeying scheme.

11.3 DISTRIBUTED ASPECTS

OSI in general is oriented towards point-to-point communication. Are the distributed aspects of key management within the scope of OSI? The ECMA model [ECMA TR/46] describes different facilities, for instance, an Authentication Facility. How could these facilities be used for key management? In the ECMA model, it is the Interdomain Facility´s responsibility to do the needed translations so that interdomain communication can occur. Can this facility solve the problems mentioned earlier in this paper when the parties have to agree on the algorithms to use through algorithm negotiations? The communication in a distributed environment is not necessarily point-to-point. How will this affect key management?

11.4 ACCESS CONTROL

The access control aspects of key management have not been considered in this paper. Neither, do the following key management standards or proposals deal with these aspects: [ANSI X9.17], [ANSI X9.28], [ISO 11166], [SDNS 90-4262] and [IEEE P802.10C]. However, in SDNS there is a special access control document [SDNS 90-4259]. Presumably for the other documents the thought is that it is up to the local security policy to decide how to perform the access control. Access control is tied more closely to the implementation aspect of key management protocol than to the protocol specification itself. ISO SC 21 has ongoing activities to develop a framework for access control [ISO 10181-3]. [ISO 10164] specifies objects and attributes for access control for OSI Management applications. It does not define an access control policy for management applications. Is it possible to make a generic access control document for key management that can be customized by the protocol user? Or, is the use of a key management protocol tightly coupled to a use of a specific security protocol, so that access control must take both into consideration? [SDNS 90-4259] indicates the latter.

11.5 ORANGE BOOK ASPECTS

Trusted Computer System Evaluation Criteria defined in the Orange Book [DoD 5200.28] is outside the scope of OSI key management protocols, but is an important consideration for an OSI key management implementation. Should the key management stack be separated from the user stack and perhaps be implemented in a black box? What level of trust, according to the orange book, should there be on a system employing key management and security protocols? Different organizations, for example, military and commercial, may have different demands. This issue is further highlighted in the article Network Security: The Parts Of The Sum [WALK89].

11.6 ZERO KNOWLEDGE TECHNIQUES

ISO SC 27 has ongoing activities on standardizing security services based on zero knowledge techniques. Can zero knowledge techniques be used for key management purposes? If so, from a protocol designers point of view, how would zero knowledge techniques affect the protocol? It has been mentioned that zero knowledge techniques require long handshaking procedures. Perhaps a four-way handshake procedure, as described in Annex B, will not support such techniques.

11.7 SECURITY MANAGEMENT

How can security management support key management? The question may seem a little bit strange because key management according to [ISO 7498-2] is part of security management. Some propositions are:

- reconfiguration of default algorithms;
- alteration of cryptographic variables;
- certificate distribution (instead of rekeying in the key management protocol).

Key management will be used to create keys so that security management information can be transferred safely.

11.8 KERBEROS

The Kerberos Authentication System [STEI88] was developed as part of MIT´s Project Athena. It is a refinement of the (trusted third-party authentication) scheme defined in [NEED78]. Since Kerberos is beginning to gain commercial support, this paper should have included a description.

ANNEX A KEY MANAGEMENT APPROACHES

This informative annex is intended for the readers who are unfamiliar with the techniques used to exchange keys automatically. It also introduces the components needed to make a key exchange possible.

There are, at the moment, two alternative techniques for performing a key exchange:

- Symmetric techniques;
- Asymmetric techniques.

In the following discussions [ANSI X9.17] is chosen as an example of symmetric form of key management, and Diffie-Hellman is chosen as an example of an asymmetric form of key management. The Diffie-Hellman algorithm is described in the article The First Ten Years Of Public Key Cryptography [DIFF88].

A1 SYMMETRIC FORM OF KEY MANAGEMENT

Traditionally, in a symmetric form of key management, the subscribers share a Key Encrypting Key (KEK) with a Key Distribution Center (CKD). Maybe the abbreviation KDC is more often used in published papers, for most of this paper CKD is used because it is the abbreviation used in [ANSI X9.17]. In most cases one subscriber does not share a KEK with every other subscriber because of the enormous number of KEKs required.

$$\frac{n * (n - 1)}{2}$$

would be needed, where n is the number of subscribers. This has already been mentioned in section 5.3. The KEK is used to encrypt other keys, the encrypted keys can either be data keys (KDs) or other KEKs. If the encrypted keys are other KEKs, the automatically distributed KEKs are then used to encrypt the data keys. A KEK is only allowed to encrypt keys and cannot be used to encrypt data. Logically, it follows that the data keys can only be used to encrypt data and cannot be used to encrypt other keys. However, it is not entirely correct to say that the KEK is used to encrypt other keys since in [ANSI X9.17] and [ANSI X9.28] offset encryption and notarization are employed. [ANSI X9.17] defines notarization as:

> A method of applying additional security to a key utilizing the identities of the originator and the ultimate recipient.

Thus, when notarization or just offset encryption is done, the KEK is transformed. Before notarization is explained, a double DES encryption must be described . The following figure is taken from Security For Computer Networks [DAVI89]. Two single DES keys are concatenated to form a double DES key. The double DES key consists of two halves, KKl and KKr respectively.

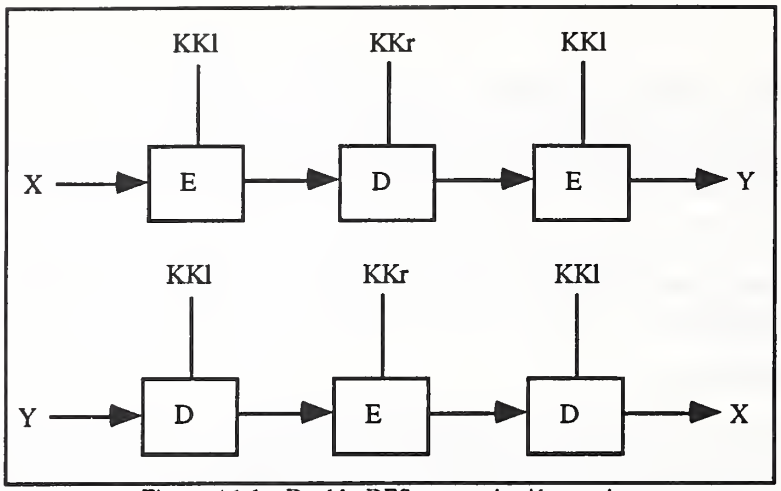


Figure A1:1 Double DES encryption/decryption

Encryption is performed by encrypting the plaintext X with KKl, thereafter the result of the encryption is decrypted with KKr. The decryption is followed by a new encryption with KKl. Double DES decryption is the inverse operation, see the figure above. The motive to employ a double DES key is that the KEK becomes cryptographically stronger than a single DES key. More about double DES encryption can be found in [MEYE82a]. Now notarization can be explained. The following figure is also taken from [DAVI89] and explains notarization when using a double DES key as a KEK. As explained above the double DES key consists of two parts, KKl and KKr. The identity of the initiating entity is given by FO and the identity of the responding entity is given by TO. The identities are split into two halves. The identity of the initiating entity, FM, is split into FM1 and FM2 respectively. The identity of the responding entity, TO, is split into TO1 and TO2 respectively. CT is a counter which is associated with the *KK.

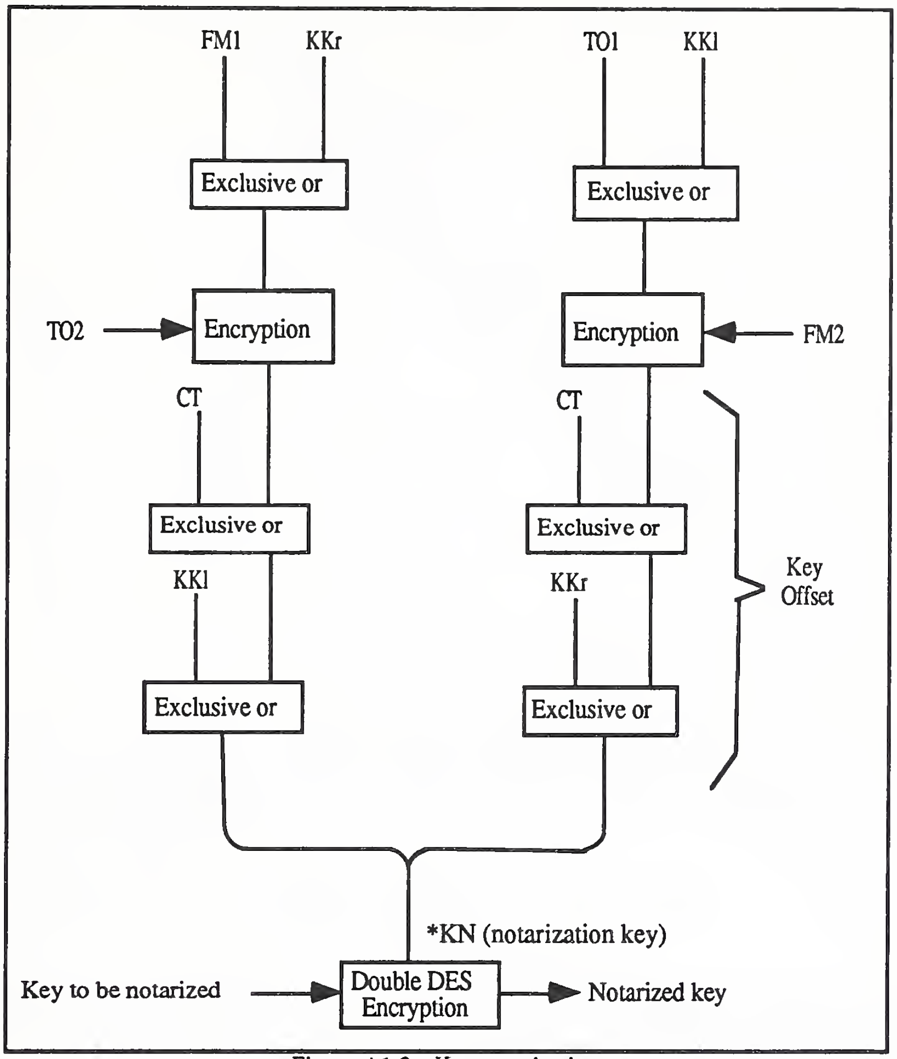


Figure A1:2 Key notarization

A notarization key, *KN, is produced, a data key is encrypted with *KN. The data key then becomes notarized. By including the identities of the originator and recipient the situation is avoided where a legitimate party can perform a masquerade. The article An Analysis Of The ISO 8732 - Key Management Standard [ZANO90] lists a number of attacks that would be possible when key notarization is not employed. One of the attacks consists of Party C masquerading as Party A. Party C requests from the CKD a data key, KD, to be shared between Party C and Party B. The CKD generates the KD and delivers it to Party C. One copy is encrypted with the *KK shared between CKD and Party C, i.e., *KKc. The other copy is encrypted with the *KK shared between CKD and B, i.e., *KKb. Party C after receiving the KD generates a fake KSM where Party C impersonates Party A.

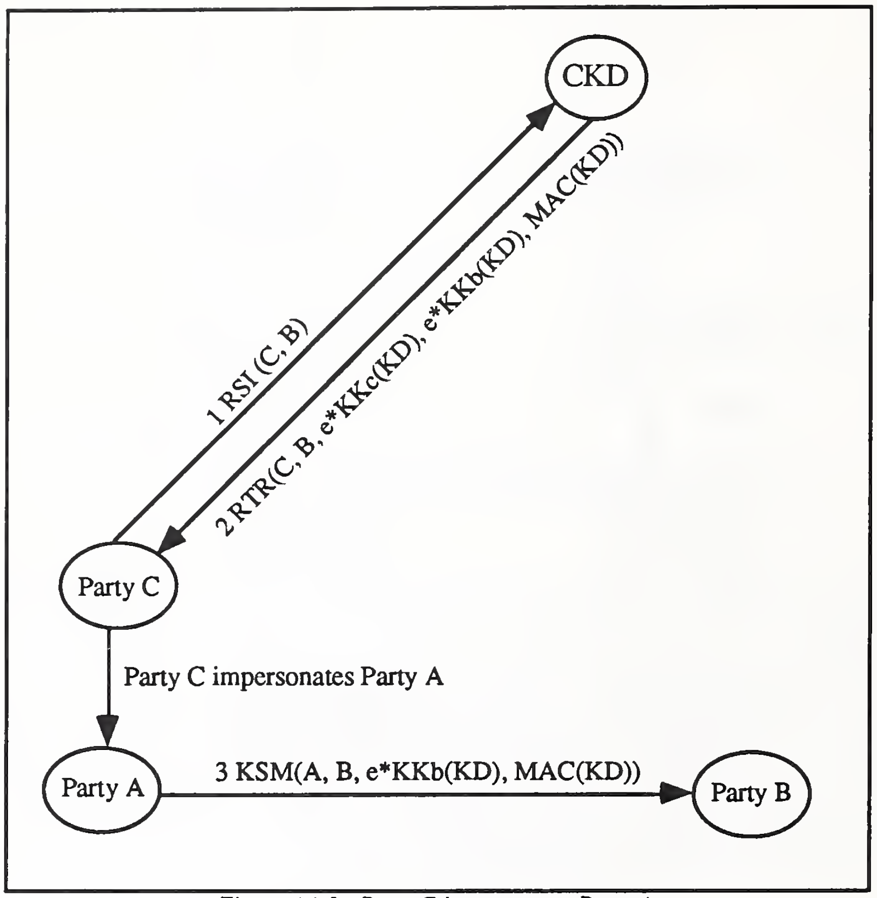

CKD
1 RSI (C, B)
2 RTR(C, B, e*KKc(KD), e*KKb(KD), MAC(KD))
Party C
Party C impersonates Party A
Party A
3 KSM(A, B, e*KKb(KD), MAC(KD))
Party B

Figure A1:3 Party C impersonates Party A

However, when notarization is employed, Party B´s computed MAC-value differs from the one included in the KSM, since Party B after having extracted the KD from KSM obtains a garbled KD. This is because Party B when extracting the KD used the identity of Party A and not the identity of Party C to produce the notarization key.

The following figure and explanation follows closely an example given in Discussions Of Symmetrical (Secret Key) And Asymmetrical (Public Key) Cryptography [BARK91] and gives a more detailed explanation of how two [ANSI X9.17] parties exchange data keys (KDs) compared to the example appearing in section 5.3.1.

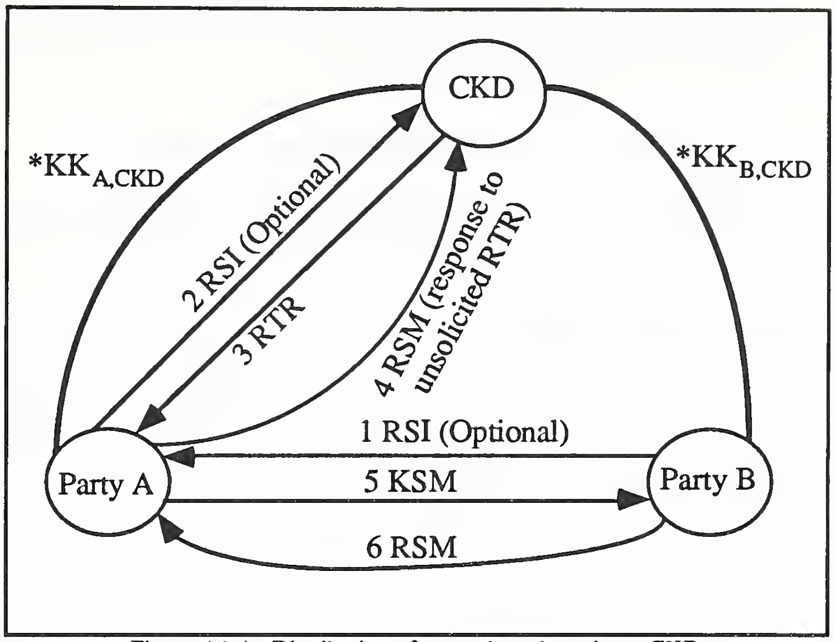


Figure A1:4 Distribution of secret keys by using a CKD

A Key Distribution Center (CKD) exists for the purpose of generating data keys (KDs) and distributing them to the CKD´s subscribers who cannot generate their own keys. Keys are distributed by using three, four or five Cryptographic Service Messages (CSMs). It is possible for the CKD, Party A or Party B to initiate the key exchange. However, the normal case is to let Party A initiate the key exchange.

- In this example one of the subscribers (Party A) sends a Request Service Initiation (RSI) to the CKD requesting one or two KDs (and optionally, an initialization vector).
- The CKD generates the requested keys and sends a Response to Request (RTR) message containing two copies of the requested keys. One copy (Party A's copy) is encrypted using the notarizing key derived from the manually distributed *KK between Party A and the CKD, that is, $*KK_{A,CKD}$; the other copy (Party B´s copy) is encrypted using the notarizing derived from the manually distributed *KK shared between Party B and the CKD, that is, $*KK_{B,CKD}$. Integrity protection for the message is provided by the presence of a Message Authentication Code (MAC) which is calculated using the KDs sent in the message. In an OSI environment the term Integrity Check Value (ICV) would be used instead of MAC.
- Party A sends Party B's copy of the keying material to Party B in a KSM. Again, integrity protection for the message is provided by the presence of a MAC which is calculated using the KDs sent in the message.
- If the KSM is received correctly Party B responds with a Response Service Message (RSM). The RSM is integrity protected with a MAC, which is computed using the KDs previously distributed by the KSM.

The key used to create the MAC is produced by an exclusive-or of the two data keys included in the KSM. If only one data key is included in the KSM that key is used to produce the MAC. One method to compute the MAC is by performing DES encryption in cbc mode, the last four bytes of the ciphertext are used as the MAC. The procedure is described in [ANSI X9.9]. Replay detection consists of a counter associated with the KEK. If someone replays a previously sent CSM, the receiving party detects that the counter has a lower value than expected and discards therefore the CSM. By having offset the keys, in a CKD environment the situation is avoided where Party A sends to Party B a key already sent in a previous KSM (belonging to another transaction) without Party B being able to detect that.

A2 ASYMMETRIC FORM OF KEY MANAGEMENT

A more detailed information about the algorithm *Diffie-Hellman*, used in this example, is given in [DIFF88].

Dennis and Russ are the two parties performing a Diffie-Hellman key exchange. Dennis begins by generating a secret X and sends to Russ:

$\alpha^X \bmod \rho$.

Russ now generates a secret Y and sends to Dennis:

$\alpha^Y \bmod \rho$.

Both Dennis and Russ can now derive the number:

$\alpha^{XY} \bmod \rho$.

Dennis derives the number by raising the data received from Russ to the exponent X, which is the secret value:

$(\alpha^Y \bmod \rho)^X \bmod \rho = \alpha^{XY} \bmod \rho$.

Similarly Russ can derive the number by raising the data received from Dennis to the exponent Y:

$(\alpha^X \bmod \rho)^Y \bmod \rho = \alpha^{XY} \bmod \rho$.

The two communicating parties then share a common number:

$\alpha^{XY} \bmod \rho$.

The number can be seen as a long bit string where a part of the string is used as the traffic encryption keys. A wiretapper can only observe the data sent from Dennis to Russ and from Russ to Dennis, but the wiretapper cannot determine the secret value of X or Y. The wiretapper will therefore not be able to form the number:

$\alpha^{XY} \bmod \rho$.

The recommendations given in Discussions Of Symmetrical (Secret Key) And Asymmetrical (Public Key) Cryptography [BARK91] is to have the number r as a large prime number with r-1 as a large prime factor. The number a must be in the range $1<a<r-1$. According to the key exchange procedure explained in the NLSP document [ISO 11577] the numbers a and r are part of the key exchange, as shown below:

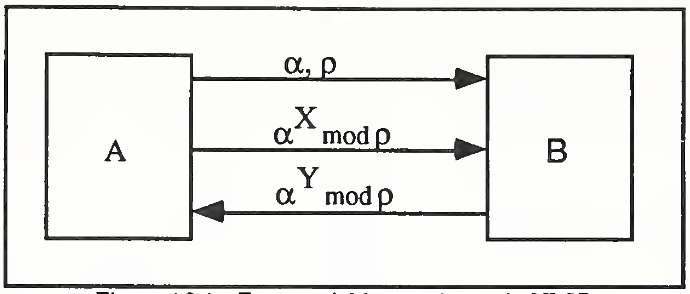


Figure A2:1 Exponential key exchange in NLSP

[BARK91] mentions that r should be fixed because it is too time consuming to generate a strong prime number every time traffic encryption keys are formed. The publication Public Key Cryptography [NECH91] gives more information about how to choose these numbers.

Diffie-Hellman has no built-in authentication. Something must be included in the above example so the two parties can be sure of each others identity. The article Key Management Systems Combining X9.17 And Public Key Cryptography [GRAFF90] gives a description of how to perform the authentication. The involved parties need to have what [GRAFF90] calls a *long term key*[1]. The public part of the key pair can be included in a *certificate*. The certificate must be signed by a trusted authority, the *Certification Authority (CA)*. The CA vouches for the binding between the distinguished name of the user and the public key component, both are included in the certificate.

In the OSI community the natural place to store the certificates of human users or system entities is in the OSI Directory [ISO 9594-1]-[ISO 9594-8]. The Directory Authentication Framework [ISO 9594-8] contains the following ASN.1 [ISO 8824] certificate definition:

```
Certificate ::= SIGNED SEQUENCE {
                version [0]             Version DEFAULT v1988,
                serialNumber            CertificateSerialNumber,
                signature               AlgorithmIdentifier,
                issuer                  Name,
                validity                Validity,
                subject                 Name,
                subjectPublicKeyInfo    SubjectPublicKeyInfo}
```

Some of the fields included in the certificate are:

issuer, the distinguished name of the Certification Authority that is the issuer of the certificate;

validity, during which period of time the certificate is valid;

subject, the distinguished name of the certificate holder;

subjectPublicKeyInfo, the value of the public key.

SIGNED is an ASN.1 macro which describes the certificate signing procedure:

[1] Can be either a RSA or ElGamal private and public key pair. [BARK91] provides more information about both RSA and ElGamal.

```
SIGNED MACRO      ::=
BEGIN
TYPE NOTATION     ::= type (ToBeSigned)
VALUE NOTATION  ::= value (VALUE
    SEQUENCE {
      ToBeSigned,
      AlgorithmIdentifier,
      ENCRYPTED OCTET STRING }
END -- of SIGNATURE
```

ToBeSigned is the certificate information needed to be signed by a certification authority. If we study the above ASN.1 certificate declaration, the certificate information consists of what is included in the SEQUENCE declaration:

```
SEQUENCE {
   version [0]...
       •
       •
       •
   ...SubjectPublicKeyInfo}.
```

AlgorithmIdentifier identifies the algorithm used to sign the certificate. *ENCRYPTED OCTET STRING* represents the encryption with the private key of the Certification Authority on a hash value computed over the certificate information; ENCRYPTED OCTET STRING is a digital signature.

The descriptions of Certificate and Certification Authority given in [ISO 9594-8] are:

> *user certificate (certificate)*: The public keys of a user, together with some other information, rendered unforgeable by encipherment with the secret[2] key of the certification authority.
>
> *certification authority*: An authority trusted by one or more users to create and assign certificates.

Using certificates, we are able to perform a Diffie-Hellman exchange which includes authentication information. We assume that the certificates of Dennis and Russ are signed by the same CA and that both Dennis and Russ know CA´s public key:

Dennis sends to Russ:

Certificate of Dennis Branstad
$\alpha^{X} \bmod \rho$
Digital signature computed on $\alpha^{XY} \bmod \rho$

Russ is able to check the correctness of the certificate. This by checking whether the signature on the certificate is correct (by using the public key of the CA), and that the certificate has not expired (by checking the validity fields). If the certificate is in order, Russ extracts the public key of

[2] Once again, in this paper the term private key is used.

Dennis from the certificate. With Dennis´ public key, Russ checks the correctness of the digital signature computed on the number $\alpha^X \mod \rho$. If the digital signature can be verified, Russ is convinced that the message originates from Dennis.

Russ then sends the following to Dennis:

Certificate of Russell Housley
$\alpha^Y \mod \rho$
Digital signature computed on $\alpha^{XY} \mod \rho$

The previously described procedure is now repeated by Dennis.

There are no precautions taken against replay in the above exchanges. The thought is that a party performing a replay attack is not in possession of either X or Y. The attacker can only replay one of the numbers: $\alpha^X \mod \rho$ or $\alpha^Y \mod \rho$. For instance, suppose the attacker, Malicious, is replaying data sent by Dennis to Russ. The probability that Russ will generate Y again is very small. Suppose instead that Russ generates Z. Malicious is therefore not able to replay the data sent in the previous session. The data sent in the previous session was protected by a traffic encryption key formed from the secret values X and Y. The data sent by Malicious is therefore not understandable to Russ, because Russ expects the traffic encryption key to be formed by X and Z. The reason why Malicious cannot form a valid traffic encryption key is because Malicious is not in possession of the secret value X. The proposed key management protocol of Annex B encrypts the Ssrq and Ssrs PDUs with the formed traffic encryption key. The replayer Malicious would not be able to correctly encrypt or decrypt those PDUs, which Russ would discover.

It should be mentioned that Appendix C of [ISO 11577] suggests the signature to be computed on the string:

$\alpha^{XY} \mod \rho$.

The signature has to be computed by both parties. The protocol of Annex B allows the Ssrq and Ssrs PDUs to carry authentication information. Thus, a key exchange according to Appendix C of NLSP can be performed.

A3 COMPARISONS

In this section comparisons are made between an asymmetric and symmetric form of key management. The advantages listed of the asymmetric form of key management are taken from The First Ten Years Of Public Key Cryptography [DIFF88] and Discussions On Draft Proposal For Public-Key Options For Key Management [OMUR89].

The main advantage of symmetric key management over asymmetric key management is the computational speed. Symmetric algorithms are much faster than asymmetric ones.

The advantages of asymmetric form of key management over symmetric are:

- The asymmetric form of key management is not based on sharing secret information. According to figure A1:4, when an [ANSI X9.17] (symmetric) key exchange is performed, the parties must share a KEK with the CKD. The highest key in the hierarchy must always be distributed manually, for example, by trusted couriers. Commercial organizations are not as good as the military at maintaining such secrets. The military has long established routines for manually distributing KEKs. Also, manual distribution is costly.
- In a [ANSI X9.17] CKD environment the CKD must be involved in every key exchange occurring in the network. Involving the CKD every time an exchange occurs causes overhead. This overhead can be avoided in asymmetric key management.
- The security of the whole network depends on the CKD/CKT. If the CKD/CKT is compromised, the whole security of the network collapses.
- The Diffie-Hellman values X and Y are unique for every session. If either X or Y is compromised, it does not affect previous or future sessions. However, in a symmetric form of key management, if the highest key in hierarchy becomes compromised, all previously exchanged data keys and future exchanged data keys, dependable of the KEK, are jeopardized. If the Diffie-Hellman algorithm is used and the long term key becomes compromised, the unauthorized possessor can perform a masquerade, but can not gain anything by passively monitoring the communication.

ANNEX B PROTOCOL AND SERVICE DEFINITIONS

This annex contains a partial specification of key management protocol and service definitions. Only the KCA-KCA protocol is omitted and some other items related to the protocol definitions given, such as labels. As previously mentioned, the ASN.1 definitions focus only on asymmetric based key exchange procedures. Note that {iso organisation security(1) modules(1)} is just a fictitious declaration.

B1 PROTOCOL DEFINITIONS

Part B1 contains the ASN.1 definitions of the various modules.

B1.1 SECURITY DEFINITIONS MODULE

```
SecurityDefinitions {iso organisation security(1) modules(1) securityDefinitions(2)}

DEFINITIONS ::=
BEGIN

EXPORTS
    SEALED, ENCRYPTED;


SEALED MACRO        ::=
BEGIN
TYPE NOTATION              ::= type(ToBeProtected)
VALUE NOTATION             ::= value(VALUE
    SEQUENCE {
        SEQUENCE {
          ToBeProtected,
          padding OCTET STRING OPTIONAL},
        icv OCTET STRING
        -- where the OCTET STRING is icv size long and obtained by applying
        -- an integrity algorithm on SEQUENCE {ToBeProtected and padding}
        })
END -- of SEALED


ENCRYPTED MACRO            ::=
BEGIN
TYPE NOTATION              ::= type(ToBeEnciphered)
VALUE NOTATION             ::= value (VALUE BIT STRING)
END -- of ENCRYPTED
END -- of SecurityDefinitions
```

B1.2 SECURITY REGISTER MODULES

```
SecurityRegister {iso organisation security(1) modules(1) securityRegister(1)}

DEFINITIONS ::=
BEGIN

EXPORTS
    keyExchMethods, integrityAlgs, confidentialityAlgs, securityLevels,
    securityLabels, signatureAlgs, rekeyingMethods;

-- Dummy definition.
IMPORTS
    security
        FROM Register {iso organisation security(1) modules(1)};


keyExchMethods        OBJECT IDENTIFIER        ::= {security 1}
integrityAlgs         OBJECT IDENTIFIER        ::= {security 2}
confidentialityAlgs   OBJECT IDENTIFIER        ::= {security 3}
securityLevels        OBJECT IDENTIFIER        ::= {security 4}
securityLabels        OBJECT IDENTIFIER        ::= {security 5}
signatureAlgs         OBJECT IDENTIFIER        ::= {security 6}
rekeyingMethods       OBJECT IDENTIFIER        ::= {security 7}

END -- of SecurityRegister
```

B1.2.1 Confidentiality Algorithms

```
ConfidentialityAlgs
        {iso organisation security(1) modules(1) securityRegister(1) confidentialityAlgs(1)}

DEFINITIONS ::=
BEGIN

EXPORTS
    symmetricAlgs, asymmetricAlgs;

IMPORTS
    confidentialityAlgs
        FROM SecurityRegister
          {iso organisation security(1) modules(1) securityRegister(1)};


symmetricAlgs ::= {confidentialityAlgs 1}
asymmetricAlgs ::= {confidentialityAlgs 2}

END -- of ConfidentialityAlgs
```

B1.2.1.1 Symmetric algorithms

```
SymmetricAlgs
        {iso organisation security(1) modules(1) securityRegister(1) confidentialityAlgs(1) symmetricAlgs(1)}

DEFINITIONS ::=
BEGIN

IMPORTS
      ALGORITHM
           FROM AuthenticationFramework
                     {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)}

      symmetricAlgs
           From {iso organisation security(1) modules(1) securityRegister(1) confidentialityAlgs(1)};

des ALGORITHM
        PARAMETER SEQUENCE {
                              mode INTEGER{ecb(0), cbc(1), cfb(2)},
                              initVector OCTET STRING OPTIONAL}
        ::= {symmetricAlgs 1}


END -- of SymmetricAlgs
```

B1.2.1.2 Asymmetric algorithms

```
AsymmetricAlgs
        {iso organisation security(1) modules(1) securityRegister(1) confidentialityAlgs(1) asymmetricAlgs(2)}


DEFINITIONS ::=
BEGIN

IMPORTS
    ALGORITHM
        FROM AuthenticationFramework
                {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)}

    asymmetricAlgs
        FROM ConfidentialityAlgs
            {iso organisation security(1) modules(1) securityRegister(1) confidentialityAlgs(1)};


-- Copied from the Directory Authentication Framework.
rsa    ALGORITHM
       PARAMETER KeySize
::= {asymmetricAlgs 1}

KeySize ::= INTEGER

END -- of AsymmetricAlgs
```

B1.2.2 Key Exchange Methods

```
KeyExchangeMethods
        {iso organisation security(1) modules(1) securityRegister(1) keyExhangeMethods(3)}

DEFINITIONS ::=
BEGIN

IMPORTS
      SIGNED, ALGORITHM, CertificationPath
           FROM AuthenticationFramework
                     {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)}

    keyExchMethods
         FROM SecurityRegister
                   {iso organisation security(1) modules(1) securityRegister(1)}

   ENCRYPTED
           FROM SecurityDefinitions
               {iso organisation security(1) modules(1) securityDefinitions(1)};


ALGORITHM-ARGUMENT-REGISTRATION MACRO    ::=
BEGIN
TYPE NOTATION     ::= "ARGUMENT" type
VALUE NOTATION             ::= value (VALUE OBJECT IDENTIFIER)
END -- of ALGORITHM-ARGUMENT-REGISTRATION


EXPONENTIATION MACRO          ::=
BEGIN
TYPE NOTATION              ::= type(Alpha) type(Rho) type(Random)
VALUE NOTATION             ::= value (VALUE INTEGER)
 -- the value of the INTEGER is generated by exponentiation
 -- of Alpha with Random mod Rho, i.e., (a  ** random) mod p
END -- of EXPONENTIATION


-- Definition of Diffie-Hellman constants a (alpha) and r (rho).

A ::= INTEGER
P ::= INTEGER

-- Definition of random numbers.

RandomA ::= INTEGER
RandomB ::= INTEGER
```

```
-- Registration of Diffie-Hellman with the authentication performed during
-- the key establishment phase.

diffieHellmanMode1 ALGORITHM
      PARAMETER SEQUENCE {
                              A,
                              P}
      ::= {keyExchMethods 1}


-- Information to be present in the Nkrq.

diffieHellmanMode1NkrqData ALGORITHM-ARGUMENT-REGISTRATION
       ARGUMENT SEQUENCE {
                  certPath CertificationPath,
                  unidirectKeys BOOLEAN OPTIONAL,
                  SIGNED EXPONENTIATION A P RandomA}
::= {keyExchMethods 1 1}


-- Information to be present in the Nkrs.

diffieHellmanMode1NkrsData ALGORITHM-ARGUMENT-REGISTRATION
      ARGUMENT SEQUENCE {
                  certPath CertificationPath,
                  unidirectKeys BOOLEAN OPTIONAL,
                  SIGNED EXPONENTIATION A P RandomB}
::= {keyExchMethods 1 2}


-- Registration of the RSA based key exchange algorithm.

rsaMode1 ALGORITHM
      PARAMETER TekSize
::= {keyExchMethods 2}


-- Information to be present in the Nkrq.

rsaMode1NkrqData ALGORITHM-ARGUMENT-REGISTRATION
      ARGUMENT SEQUENCE {
                  certPath CertificationPath,
                  unidirectKeys BOOLEAN OPTIONAL,
                  ENCRYPTED RandomA}
::= {keyExchMethods 2 1}


-- Information to be present in the Nkrs.

rsaMode1NkrsData ALGORITHM-ARGUMENT-REGISTRATION
      ARGUMENT SEQUENCE {
                  certPath CertificationPath,
                  unidirectKeys BOOLEAN OPTIONAL,
                  ENCRYPTED ENCRYPTED RandomB}
::= {keyExchMethods 2 2}
```

```
TekSize ::= INTEGER

END -- KeyExchangeMethods
```

This is another way to do the registration by using an ASN.1 macro. In the TYPE NOTATION field of the macro is specified the information to be present in a specific Kmpdu.

```
KEY-EXCHG-ALG MACRO ::=
BEGIN
   TYPE NOTATION ::=    ParameterData NkrqData NkrsData SsrqData
                        SsrsData RkrqData RkrsData RkdrqData RkdrsData
   VALUE NOTATION ::= VALUE(value OBJECT IDENTIFIER)
   ParameterData ::= empty | "PARAMETER" type
   NkrqData     ::= "NkrqData" type
   NkrsData     ::= "NkrsData" type
   SsrqData     ::= empty    | "SsrqData"     type
   SsrsData     ::= empty    | "SsrsData"     type
   RkrqData     ::= empty    | "RkrqData"     type
   RkrsData     ::= empty    | "RkrsData"     type
   RkdrqData    ::= empty    | "RkdrqData"    type
   RkdrsData    ::= empty    | "RkdrsData"    type
END


-- Registration of Diffie-Hellman with the authentication
-- performed during the key establishment phase.

diffieHellmanMode1 KEY-EXCHG-ALG
   PARAMETER SEQUENCE {
                        alpha INTEGER,
                        rho INTEGER}
   NkrqData SEQUENCE {
                        certPath CertificationPath,
                        unidirectKeys BOOLEAN OPTIONAL,
                        SIGNED EXPONENTIATION A P RandomA}
   NkrsData SEQUENCE {
                        certPath CertificationPath,
                        unidirectKeys BOOLEAN OPTIONAL,
                        SIGNED EXPONENTIATION A P RandomB}
::= {keyExchMethod 1}
```

```
-- Registration of the RSA based key exchange algorithm.

rsaMode1 KEY-EXCHG-ALG
    PARAMETER TekSize
    NkrqData SEQUENCE {
                        certPath CertificationPath,
                        unidirectKeys BOOLEAN OPTIONAL,
                        ENCRYPTED RandomA}
    NkrsData SEQUENCE {
                        certPath CertificationPath,
                        unidirectKeys BOOLEAN OPTIONAL,
                        ENCRYPTED ENCRYPTED RandomB}
::= {keyExchMethod 2}
```

B1.2.3 Integrity Mechanisms

```
IntegrityAlgs
    {iso organisation security(1) modules(1) securityRegister(1) integrityAlgs(2)}

DEFINITIONS ::=
BEGIN

IMPORTS
   integrityAlgs
     FROM SecurityRegister
        {iso organisation security(1) modules(1) securityRegister(1)}


   ALGORITHM
     FROM AuthenticationFramework
        {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)};


Icv-size ::= INTEGER

x9dot9 ALGORITHM
       PARAMETER Icv-size
::= {integrityAlgs 1}


END -- of IntegrityAlgs
```

B1.2.4 Security Levels

```
SecurityLevels
        {iso organisation security(1) modules(1) securityRegister(1) securityLevels(4)}

DEFINITIONS ::=
BEGIN

IMPORTS
      securityLevels
          FROM SecurityRegister
                {iso organisation security(1) modules(1) securityRegister(1)}

-- This macro is used for registering security levels.
SEC-LEVEL-REG MACRO ::=
BEGIN
TYPE NOTATION         ::= "SECURITY LEVELS" type
VALUE NOTATION        ::= value(OBJECT IDENTIFIER)
END -- of SEC-LEVEL-REG


dodSecLevelType SEC-LEVEL-REG
        SECURITY LEVELS
            BIT STRING {secret(0), classified(1), secret(2), top-secret(3)}
::= {securityLevels 1}

END -- of SecurityLevels
```

B1.2.5 Security Labels

```
SecurityLabels
        {iso organisation security(1) modules(1) securityRegister(1) securityLabels(5)}

DEFINITIONS ::=
BEGIN

-- Problem: can security labels be given an ASN.1-representation when they
--- are used below the presentation layer?

END -- of SecurityLabels
```

B1.2.6 Signature Algorithms

```
SignatureAlgorithms
        {iso organisation security(1) modules(1) securityRegister(1) signatureAlgs(6)}

DEFINITIONS ::=
BEGIN

IMPORT
    ALGORITHM
      FROM AuthenticationFramework
         {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)}

   signatureAlgs
      FROM SecurityRegister
         {iso organisation security(1) modules(1) securityRegister(1)};


-- Copied from the Directory Authentication Framework
sqMod-n ::= ALGORITHM
   PARAMETER BlockSize
::= {signatureAlgs}

BlockSize ::= INTEGER

END -- of SignatureAlgorithms
```

B1.2.7 Rekeying Methods

```
RekeyingMethods
        {iso organisation security(1) modules(1) securityRegister(1) rekeyingMethods(7)}


DEFINITIONS ::=
BEGIN

IMPORTS
    SubjectPublicKeyInfo, Certificate, ALGORITHM
        FROM AuthenticationFramework
            {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)}

    rekeyingMethods
        FROM SecurityRegister
            {iso organisation security(1) modules(1) securityRegister(1)};


-- Rekeying method based on the principle that the user creates the public
-- and private key pair and sends the public part to the certification
-- authority for certification.  As a response the certification authority
-- returns a certificate.

rekWhenUserCreatesKey ALGORITHM
::= {rekeyingMethods 1}


-- Information to be present in the Rkrq.
rekMeth1RkrqData ALGORITHM-ARGUMENT-REGISTRATION
    ARGUMENT SubjectPublicKeyInfo
::= {rekeyingMethods 1 1}


-- Information to be present in the Rkrs.
rekMeth1RkrsData ALGORITHM-ARGUMENT-REGISTRATION
                        ARGUMENT Transaction
::= {rekeyingMethods 1 2}


-- Information to be present in the Rkdrq.
rekMeth1RkdrqData ALGORITHM-ARGUMENT-REGISTRATION
                        ARGUMENT Transaction
::= {rekeyingMethods 1 3}
```

```
-- Information to be present in the Rkdrs.
rekMeth1RkdrsData ALGORITHM-ARGUMENT-REGISTRATION
   ARGUMENT CHOICE {
                userCertificate      [0] Certificate,
                cACertificate        [1] Certificate}
::= {rekeyingMethods 1 4}


Transaction ::= INTEGER

END  -- of RekeyingMethods
```

This is the other way to do the registration by using an ASN.1 macro.

```
REKEY-REG MACRO ::=
BEGIN
   TYPE NOTATION ::=     ParameterData RkrqData RkrsData
                         RkdrqData RkdrsData
   VALUE NOTATION        ::=  VALUE(value OBJECT IDENTIFIER)
   ParameterData         ::=  empty | "PARAMETER" type
   RkrqData              ::=  "RkrqData" type
   RkrsData              ::=  "RkrsData" type
   RkdrqData             ::=  empty     | "RkdrqData" type
   RkdrsData             ::=  empty     | "RkdrsData" type
END


rekMeth1 REKEY-REG
   RkrqData SubjectPublicKeyInfo
   RkrsData Transaction
   RkdrqData Transaction
   RkdrsData CHOICE {
      userCertificate      [0]  Certificate,
      caCertificate        [1]  Certificate }
::= {rekeyingMethods 1}
```

B1.3 KEY MANAGEMENT PROTOCOL MODULES

```
KeyManagementProtocol
{iso organisation security(1) modules(1) keyManagementProtocol(3)}

DEFINITIONS
BEGIN

-- This module contains the definitions in common for the three key
-- management modules.


IMPORTS
   AlgorithmIdentifier
         FROM AuthenticationFramework
               {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)};

EXPORTS Nkrq, Nkrs, Ksrq, Ksrs, Kbrt, StateResult, SecServGrpProp, SecServGrpAcc,
ProtocolVersion;

-- Value of algArgId can either be determined from the key exchange method
-- that the parties agreed on in the Kirs PDU or it can be
-- stored in the SMIB.  Therefore algArgId does not need to be present in the
--  Nkrq and Nkrs PDUs and is declared optional.

Nkrq ::= SEQUENCE {
   algArgId            [0]   IMPLICIT OBJECT IDENTIFIER OPTIONAL,
   algArgument         [1]   IMPLICIT ANY DEFINED BY algArgId}


Nkrs ::= SEQUENCE {
   algArgId            [0]   IMPLICIT OBJECT IDENTIFIER OPTIONAL,
   algArgument         [1]   IMPLICIT ANY DEFINED BY algArgId}


-- The Ksrq and Ksrs PDUs are used when the key exchange is based
-- on a six way handshake procedure.
Ksrq ::= SEQUENCE {
   algArgId            [0]   IMPLICIT IMPLICIT OBJECT IDENTIFIER OPTIONAL,
   algArgument         [1]   IMPLICIT  ANY DEFINED BY algArgId}


Ksrs ::= SEQUENCE {
   algArgId            [0]   IMPLICIT OBJECT IDENTIFIER OPTIONAL,
   algArgument         [1]   IMPLICIT ANY DEFINED BY algArgId}
```

```
-- Abort PDU

Kbrt ::= SEQUENCE {
                stateResult StateResult,
                diagnostic Diagnostic OPTIONAL}


-- The confidentiality and integrity algorithms proposed by the initiator.

SecServGrpProp ::= SEQUENCE {
    -- Proposed confidentiality algorithms, first in sequence
    -- is the preferred one.
   confAlgs SEQUENCE OF AlgorithmIdentifier,
    -- Proposed integrity algorithms, first in sequence
    -- is the preferred one.
   icvAlgs SEQUENCE OF AlgorithmIdentifier}


-- The confidentiality and integrity algorithm accepted by the responder.

SecServGrpAcc ::= SEQUENCE {
   confAlg AlgorithmIdentifier,
   icvAlg AlgorithmIdentifier}

ProtocolVersion ::= [0] IMPLICIT BIT STRING {version-1 (0)}

StateResult ::= IMPLICIT INTEGER
                  { success (0),
                    failure (1) }

Diagnostic ::= IMPLICIT INTEGER

END  -- of KeyManagementProtocol
```

B1.3.1 KMSA TO KMSA

This section contains the KMSA-KMSA protocol definitions.

B1.3.1.1 ASN.1 declarations

```
Kmsa-Kmsa-KeyManagementProtocol
    {iso organisation security(1) modules(1) keyManagementProtocol(3)
    kmsa-kmsa-KeyManagementProtocol(1)}

DEFINITIONS ::=
BEGIN


IMPORTS
   AlgorithmIdentifier
         FROM AuthenticationFramework
              {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)}

Nkrq, Nkrs, Ksrq, Ksrs, Kbrt, SecServGrpProp, SecServGrpAcc, ProtocolVersion, StateResult,
Diagnostic
        FROM KeyManagementProtocol
              {iso organisation security(1) modules(1) keyManagementProtocol(3)}


   SEALED, ENCRYPTED
         FROM SecurityDefinitions
            {iso organisation security(1) modules(1) securityDefinitions(1)}

   TLSPParams
         FROM TLSPDefinitions
            {iso organisation security(1) modules(1) securityProtocolDef(4)
            tLSPDefinitions(1)};


Kmpdu ::= CHOICE {
                [0]  IMPLICIT    Kirq,
                [1]  IMPLICIT    Kirs,
                [2]  IMPLICIT    Nkrq,
                [3]  IMPLICIT    Nkrs,
                [4]  IMPLICIT    Kurq,
                [5]  IMPLICIT    Kurs,
                [6]  IMPLICIT    Ekpdu,
                [7]  IMPLICIT    Kbrt}
```

```
Ekpdu ::= ENCRYPTED SEALED CHOICE {
                              [0]  IMPLICIT    Ksrq,
                              [1]  IMPLICIT    Ksrs,
                              [2]  IMPLICIT    Ssrq,
                              [3]  IMPLICIT    Ssrs}


-- secServGrpProp and keyEchGrpProp are declared optional because
-- prior agreements may exist about the algorithms to use.  If prior
-- agreements do exist the pre-established information is stored in the SMIB.
-- The parties then have no need to establish the information conveyed in
-- secServGrpProp  and keyEchGrpProp.
Kirq ::= SEQUENCE {
                    protocolVersion ProtocolVersion,
                    secServGrpProp [1] IMPLICIT SecServGrpProp OPTIONAL,
                    keyEchGrpProp CHOICE {
                         [0]  IMPLICIT AsymmKeyExchGrpProp,
                         [1]  IMPLICIT SymmKeyExchGrpProp} OPTIONAL}


Kirs ::= SEQUENCE {
               stateResult StateResult,
               protocolVersion ProtocolVersion,
               secServGrpAcc [0] IMPLICIT SecServGrpAcc OPTIONAL,
               keyEchGrpAcc CHOICE {
                         [0]  IMPLICIT AsymmKeyExchGrpAcc,
                         [1]  IMPLICIT SymmKeyExchGrpAcc} OPTIONAL}


-- The Ssrq contains the security protocol attributes proposed by the initiator.
Ssrq ::= SET {
  -- If necessary algorithm dependent data has to be carried in this field.
  algDepData [0] IMPLICIT SET {
        algArgId          [0]  IMPLICIT OBJECT IDENTIFIER OPTIONAL,
        algArgument       [1]  IMPLICIT ANY DEFINED BY algArgId} OPTIONAL,
        -- security association identifier
  [1] said Said,
  -- Definitions associated with a particular security protocol.
  spParameters CHOICE {
          [0]  IMPLICIT    TLSPParams,
          [1]  IMPLICIT    NLSPParams,  -- currently not defined
          [2]  IMPLICIT    SDEParams}}  -- currently not defined


Ssrs ::= Ssrq
```

```
-- PDUs used to update a security association.

Kurq ::= SEQUENCE {
   remSaid Said,
   ENCRYPTED SEALED newLocSaid Said}

Kurs ::= SEQUENCE {
   remSaid Said,
   ENCRYPTED SEALED newLocSaid Said}


-- The key exchange methods proposed by the initiator.

AsymmKeyExchGrpProp ::= SEQUENCE {
    keyExhMethods SEQUENCE OF AlgorithmIdentifier
-- possibly more declarations have to be put in}


SymmKeyExchGrpProp ::= -- currently not defined


-- The algorithm selected by the responder.

AsymmKeyExchGrpAcc ::= SEQUENCE {
    keyExchMethod AlgorithmIdentifier
-- possibly more declarations have to be put in}


SymmKeyExchGrpAcc ::= -- currently not defined


Said ::= OCTET STRING


END  -- of Kmsa-Kmsa-KeyManagementProtocol
```

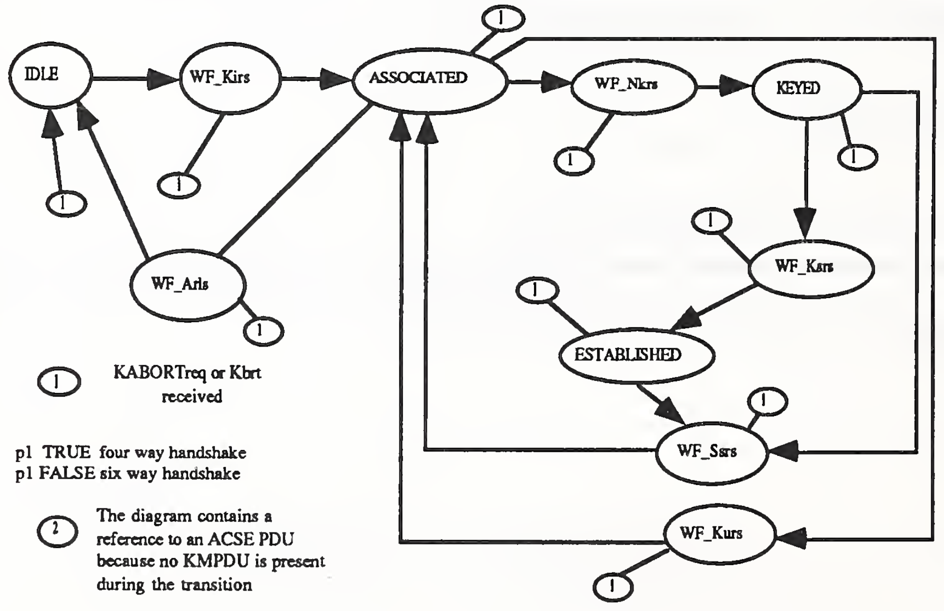
B1.3.1.2 Protocol Machine

AS INITIATOR

EVENT	NEXT STATE	CONDITION FOR TRANSITION TO OCCUR
KM-INITreq	WF_Kirs	if state = IDLE
Kirs	ASSOCIATED	if state = WF_Kirs
NEW-KEYreq	WF_Nkrs	if state = ASSOCIATED
Nkrs	KEYED	if state = WF_Nkrs
KEY-SERVreq	WF_Ksrs	if not p1 AND if state = KEYED
Ksrs	ESTABLISHED	if state = WF_Ksrs

SEC-SERVreq	WF_Ssrs	if state = KEYED AND if p1 or
		if state = ESTABLISHED AND if not p1

Ssrs	ASSOCIATED	if state = WF_Ssrs
KEY-UPDreq	WF_Kurs	if state = ASSOCIATED
Kurs	ASSOCIATED	if state = WF_Kurs
KABORTreq	IDLE	can occur in any state except IDLE
Kbrt	IDLE	can occur in any state except IDLE
KM-RELEASEreq	WF_Arls	if state = ASSOCIATED
Arls ②	IDLE	if state = WF_Arls



① KABORTreq or Kbrt received

p1 TRUE four way handshake
p1 FALSE six way handshake

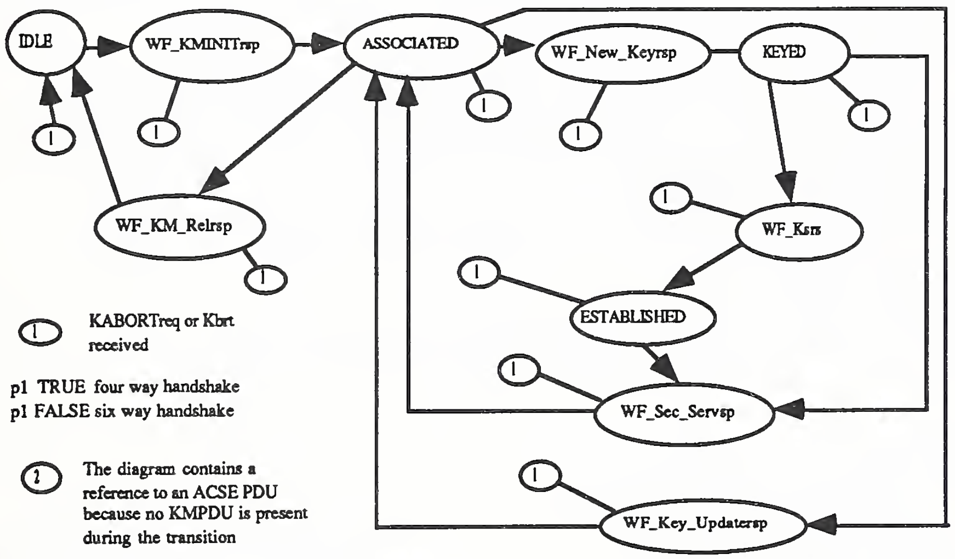
② The diagram contains a reference to an ACSE PDU because no KMPDU is present during the transition

AS RESPONDER

EVENT	NEXT STATE	CONDITION FOR TRANSITION TO OCCUR
Kirq	WF_KINITrsp	if state = IDLE
KM-INITrsp	ASSOCIATED	if state = WF_KINITrsp
Nkrq	WF_New-Keyrsp	if state = ASSOCIATED
NEW-KEYrsp	KEYED	if state = WF_New_Keyrsp
Ksrq	WF_Keyrsp	if not p1 AND if state = KEYED
KEY-SERVrsp	ESTABLISHED	if not p1 AND if state =WF_Keyrsp

Ssrq	WF_Sec-Servrsp	if state = KEYED AND p1 or
		if state = ESTABLISHED AND not p1

SEC-SERVrsp	ASSOCIATED	if state = WF_Sec_Servrsp
Kurq	WF_Key-Updatersp	if state = ASSOCIATED
KEY-UPDrsp	ASSOCIATED	if state = WF_Key-Updatersp
KABORTreq	IDLE	can occur in any state except IDLE
Kbrt	IDLE	can occur in any state except IDLE
Aarq ②	WF_KM_Relrsp	if state = ASSOCIATED
KM-RELEASErsp	IDLE	if state = WF_KM_Relrsp



① KABORTreq or Kbrt received

p1 TRUE four way handshake
p1 FALSE six way handshake

② The diagram contains a reference to an ACSE PDU because no KMPDU is present during the transition

B1.3.2 KMSA TO KCA

This section contains the KMSA-KCA protocol declarations.

B1.3.2.1 ASN.1 definitions

```
Kmsa-Kca-KeyManagementProtocol
    {iso organisation security(1) modules(1) keyManagementProtocol(3) kmsa-Kca-KeyManagementProtocol(2)}

DEFINITIONS ::=
BEGIN

IMPORTS

    AlgorithmIdentifier
        FROM AuthenticationFramework
                {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)}

  Nkrq, Nkrs, Ksrq, Ksrs, Kbrt, SecServGrpProp, SecServGrpAcc, ProtocolVersion,
  StateResult, Diagnostic
       FROM KeyManagementProtocol
              {iso organisation security(1) modules(1) keyManagementProtocol(3)}

  SEALED, ENCRYPTED
         FROM SecurityDefinitions
            {iso organisation security(1) modules(1) securityDefinitions(1)};


Kmpdu ::= CHOICE {
                [0]  IMPLICIT    Kirq,
                [1]  IMPLICIT    Kirs,
                [2]  IMPLICIT    Nkrq,
                [3]  IMPLICIT    Nkrs,
                [4]  IMPLICIT    Ekpdu,
                [5]  IMPLICIT    Kbrt}


Ekpdu ::= ENCRYPTED SEALED CHOICE {
                -- These PDUs are only valid for an asymmetric form of
                -- key management
                [0]  IMPLICIT    Rkrq,
                [1]  IMPLICIT    Rkrs
                [2]  IMPLICIT    Rkdrq,
                [3]  IMPLICIT    Rkdrs }
```

```
-- The Kirq and Kirs PDUs are used to perform the algorithm negotiation.
Kirq ::= SEQUENCE {
    protocolVersion ProtocolVersion,
    secServGrpProp [1] IMPLICIT SecServGrpProp OPTIONAL,
    keyExchGrpProp CHOICE {
                [1]  IMPLICIT AsymmKeyExchGrpProp,
                [2]  IMPLICIT SymmKeyExchGrpProp} OPTIONAL}


Kirs ::= SEQUENCE {
    stateResult ::= StateResult,
    protocolVersion ProtocolVersion,
    secServGrpAcc [0] IMPLICIT SecServGrpAcc,
    keyExchGrpAcc CHOICE {
                [1]  IMPLICIT AsymmKeyExchGrpAcc,
                [2]  IMPLICIT SymmKeyExchGrpAcc} OPTIONAL}


-- The values of the two algArgIds can be determined from
-- the Kirs PDU or SMIB.  Therefore are the algArgIds declared as OPTIONAL.
Rkrq ::= SET {
                auth        [0]  IMPLICIT SET{
                    algArgId        [0]  IMPLICIT OBJECT IDENTIFIER OPTIONAL,
                    authData        [1]  IMPLICIT ANY DEFINED BY algArgId} OPTIONAL,
                rek         [1]  IMPLICIT SET{
                    algArgId        [0]  IMPLICIT OBJECT IDENTIFIER OPTIONAL,
                    rekData         [1]  IMPLICIT ANY DEFINED BY algArgId}}


Rkrs ::= Rkrq

Rkdrq ::= Rkrq

Rkdrs ::= Rkrq


-- The algorithms proposed by the initiator.
AsymmKeyExchGrpProp ::= SEQUENCE {
    keyExchMethods              [1]  IMPLICIT SEQUENCE OF AlgorithmIdentifier,
    rekeyMethods                [2]  IMPLICIT SEQUENCE OF AlgorithmIdentifier}


SymmKeyExchGrpProp ::= -- currently not defined


-- The key exchange and rekey methods accepted by the responder.
AsymmKeyExchGrpAcc ::= SEQUENCE {
            keyExchMethod           [1]  IMPLICIT AlgorithmIdentifier,
            rekeyMethod             [2]  IMPLICIT AlgorithmIdentifier}
```

```
SymmKeyExchGrpAcc ::= -- currently not defined

END -- of Kmsa-Kca-KeyManagementProtocol
```

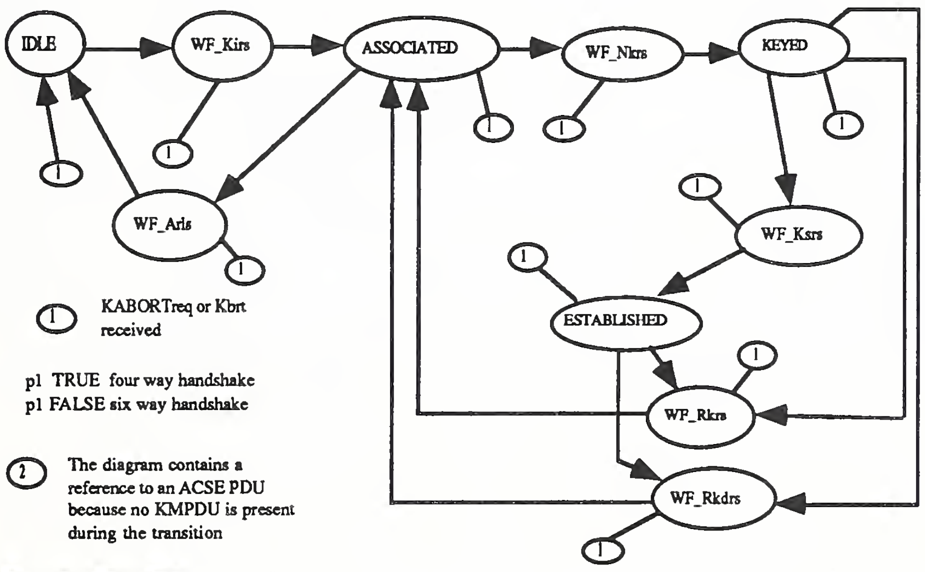
B1.3.2.2 Protocol Machine

AS INITIATOR

EVENT	NEXT STATE	CONDITION FOR TRANSITION TO OCCUR
KM-INITreq	WF_Kirs	if state = IDLE
Kirs	ASSOCIATED	if state = WF_Kirs
NEW-KEYreq	WF_Nkrs	if state = ASSOCIATED
Nkrs	KEYED	if state = WF_Nkrs
KEY-SERVreq	WF_Ksrs	if not p1 AND if state = KEYED
Ksrs	ESTABLISHED	if state = WF_Ksrs

REKEYreq	WF_Rkrs	if state = KEYED AND if p1 or
		if state = ESTABLISHED AND if not p1

Rkrs	ASSOCIATED	if state = WF_Rkrs
REKEY-DELreq	WF_Rkdrs	if state = ASSOCIATED
Rkdrs	ASSOCIATED	if state = WF_Rkdrs
KABORTreq	IDLE	can occur in any state except IDLE
Kbrt	IDLE	can occur in any state except IDLE
KM-RELEASEreq	WF_Arls	if state = ASSOCIATED
Arls ②	IDLE	if state = WF_Arls



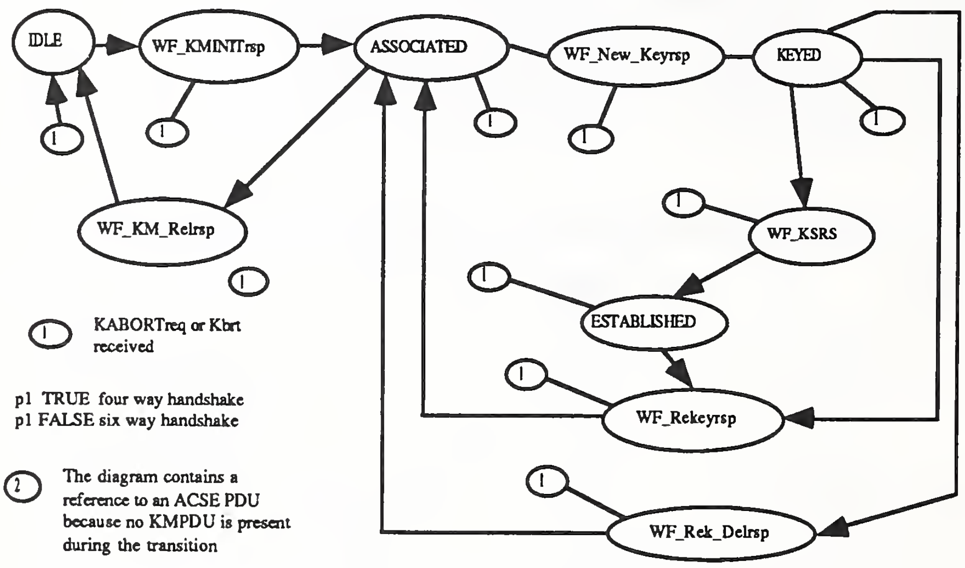
IDLE
WF_Kirs
ASSOCIATED
WF_Nkrs
KEYED
WF_Arls
WF_Ksrs
ESTABLISHED
WF_Rkrs
WF_Rkdrs
1
1 KABORTreq or Kbrt received
p1 TRUE four way handshake
p1 FALSE six way handshake
2 The diagram contains a reference to an ACSE PDU because no KMPDU is present during the transition

AS RESPONDER

EVENT	NEXT STATE	CONDITION FOR TRANSITION TO OCCUR
Kirq	WF_KINITrsp	if state = IDLE
KM-INITrsp	ASSOCIATED	if state = WF_KINITrsp
Nkrq	WF_New-Keyrsp	if state = ASSOCIATED
NEW-KEYrsp	KEYED	if state = WF_New_Keyrsp
Ksrq	WF_Keyrsp	if not p1 AND if state = KEYED
KEY-SERVrsp	ESTABLISHED	if not p1 AND if state =WF_Keyrsp

Rkrq	WF_Rekeyrsp	if state = KEYED AND p1 or
		if state = ESTABLISHED AND not p1

REKEYrsp	ASSOCIATED	if state = WF_Rekeyrsp
Rkdrq	WF_Rek_Delrsp	if state = ASSOCIATED
REKEY-DELrsp	ASSOCIATED	if state = WF_Rek_Delrsp
KABORTreq	IDLE	can occur in any state except IDLE
Kbrt	IDLE	can occur in any state except IDLE
Aarq ②	WF_KM_Relrsp	if state = ASSOCIATED
KM-RELEASErsp	IDLE	if state = WF_KM_Relrsp



① KABORTreq or Kbrt received

p1 TRUE four way handshake
p1 FALSE six way handshake

② The diagram contains a reference to an ACSE PDU because no KMPDU is present during the transition

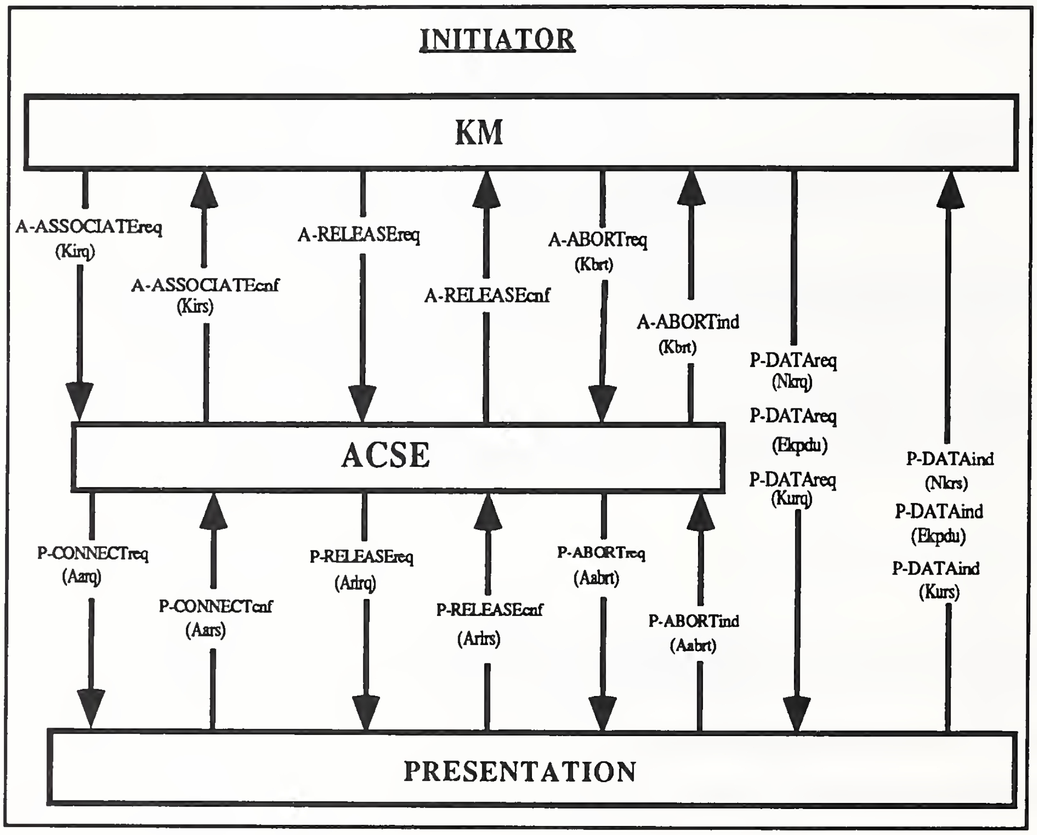
B1.3.3 KCA TO KCA

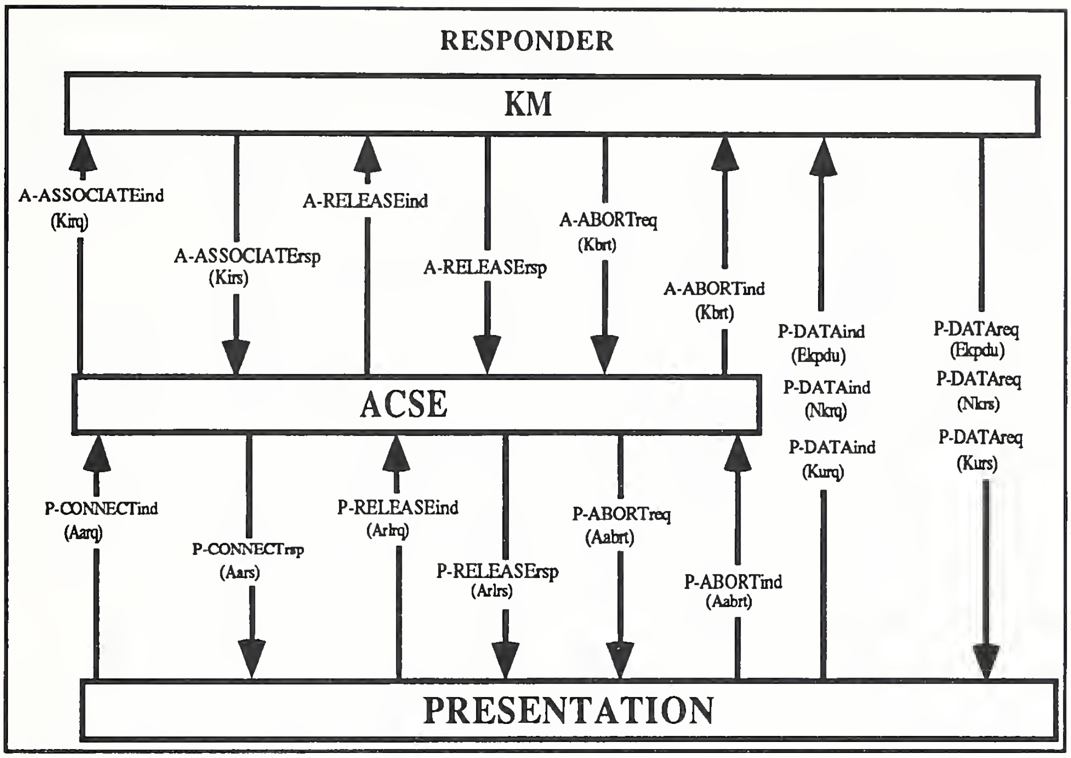
```
Kca-Kca-KeyManagementProtocol
    {iso organisation security(1) modules(1) keyManagementProtocol(3) kca-to-kca(3)}

IMPORTS

     AlgorithmIdentifier
          FROM AuthenticationFramework
                    {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)}

    Nkrq, Nkrs, Ksrq, Ksrs, SecServGrpProp, SecServGrpAcc
         FROM KeyManagementProtocol
                 {iso organisation security(1) modules(1) keyManagementProtocol(3)}

    SEALED, ENCRYPTED
           FROM SecurityDefinitions
               {iso organisation security(1) modules(1) securityDefinitions(1)};

DEFINITIONS ::=
BEGIN

-- Here should the ASN.1 definitions of the KCA-KCA protocol appear, probably
-- they will be very similar to the KMSA-KCA definitions.

END -- of Kca-Kca-KeyManagementProtocol
```

B1.4 MAPPING OF PDUs

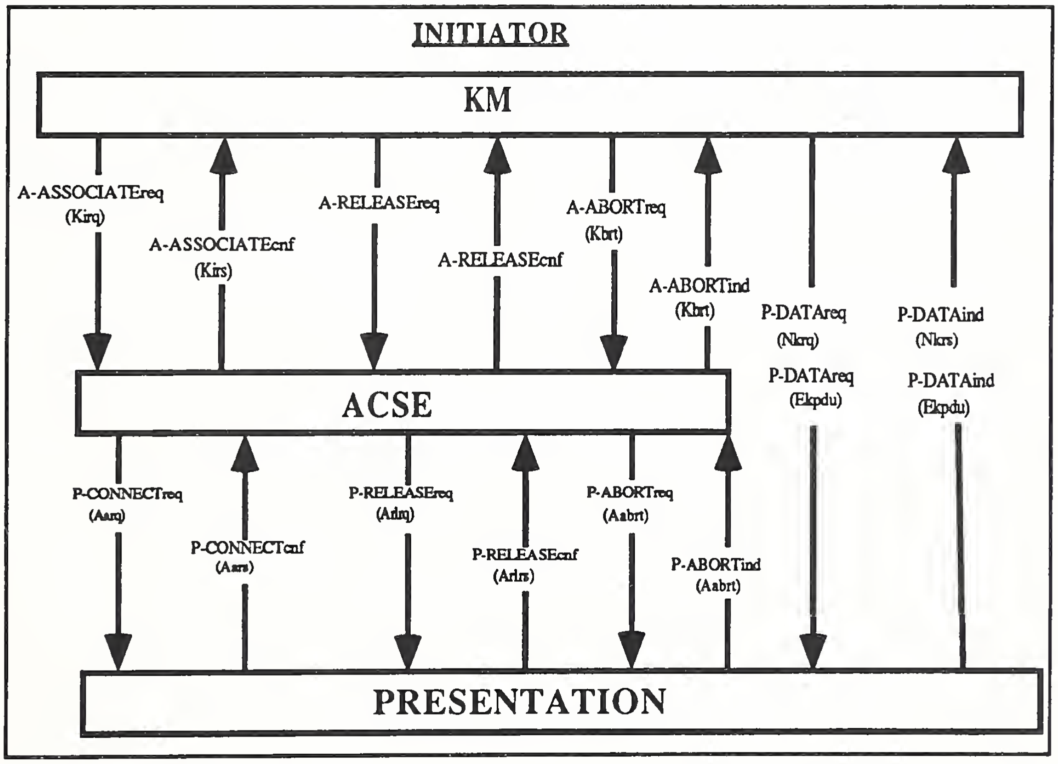
This section explains to which service primitives the key management PDUs are mapped.

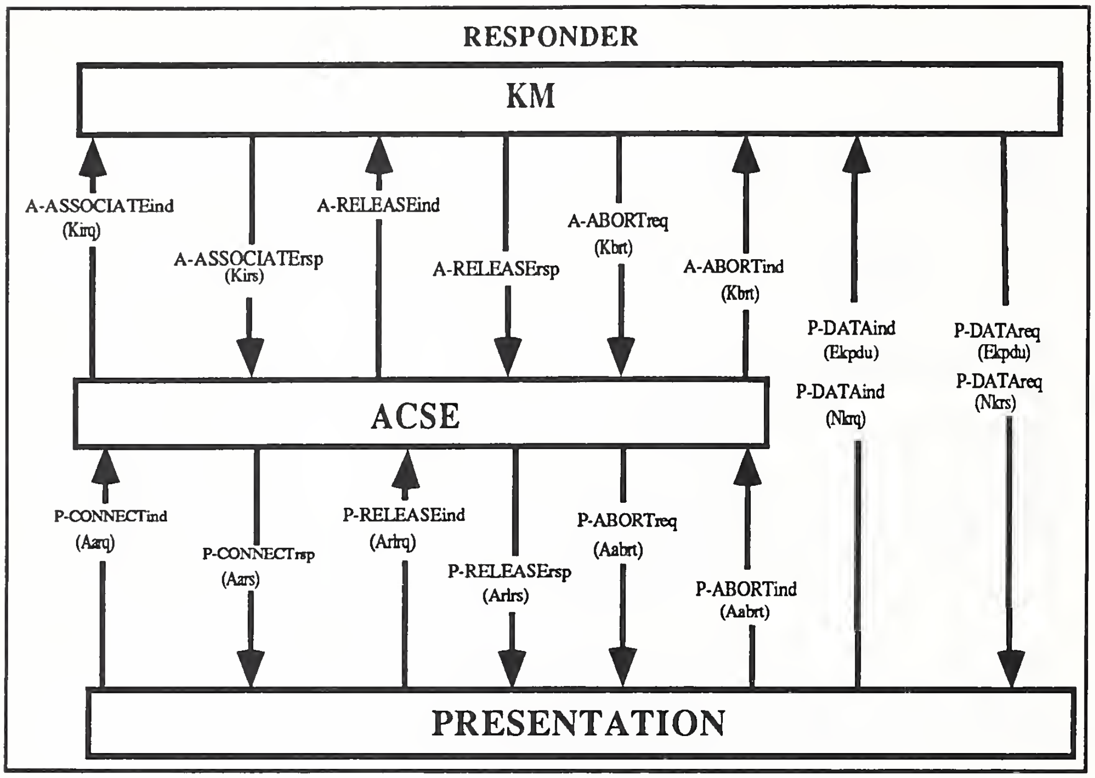
B1.4.1 KMSA TO KMSA





B1.4.2 KMSA TO KCA




RESPONDER
KM
A-ASSOCIATEind (Kirq)
A-ASSOCIATErsp (Kirs)
A-RELEASEind
A-RELEASErsp
A-ABORTreq (Kbrt)
A-ABORTind (Kbrt)
P-DATAind (Ekpdu)
P-DATAreq (Ekpdu)
ACSE
P-DATAind (Nkrq)
P-DATAreq (Nkrs)
P-CONNECTind (Aarq)
P-CONNECTrsp (Aars)
P-RELEASEind (Arlrq)
P-RELEASErsp (Arlrs)
P-ABORTreq (Aabrt)
P-ABORTind (Aabrt)
PRESENTATION

B1.4.3 KCA TO KCA

Identical to KMSA-KCA PDU mapping.

B1.5 SECURITY PROTOCOL MODULE

This section contains the ASN.1 definitions of the attributes associated with a particular security protocol.

B1.5.1 TLSP

```
TLSPDefinitions
    {iso organisation security(1) modules(1) securityProtocolDef(4) tLSPDefinitions(1)}

DEFINITIONS ::=
BEGIN

IMPORTS
    AlgorithmIdentifier
        FROM AuthenticationFramework
                {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)};


EXPORTS
    TLSPParams;

SecLevelsIdentifier ::= AlgorithmIdentifier

TLSPParams ::= SET {
        confid       [0]  IMPLICIT BOOLEAN,
        integr       [1]  IMPLICIT BOOLEAN,
        secLabelDef       [2]  CHOICE {
                            propLabels       [0]  IMPLICIT PropLabels,
                            accLabel         [1]  IMPLICIT AccLabel} OPTIONAL,
        kindOfKey INTEGER{perNSAP(0), perTConnection(1)}}


PropLabels ::= SET OF {
            secLevelSet         [0]  IMPLICIT SecLevelsIdentifier,
            secLabelType        [1]  IMPLICIT OBJECT IDENTIFIER}

AccLabel ::= SET {
            secLevelSet         [0]  IMPLICIT SecLevelsIdentifier,
            secLabelType        [1]  IMPLICIT OBJECT IDENTIFIER}

END  -- of TLSPDefinitions
```

B1.5.2 NLSP

```
NLSPDefinitions
    {iso organisation security(1) modules(1) securityProtocolDef(4) nLSPDefinitions(2)}

BEGIN

-- Should contain the attributes needed by NLSP.

END  -- of NLSPDefinitions
```

B1.5.3 SDE

```
SDEDefinitions
    {iso organisation security(1) modules(1) securityProtocolDef(4) sDEDefinitions(3)}

BEGIN

-- Should contain the attributes needed by SDE.

END -- of SDEDefinitions
```

B2 SERVICE SPECIFICATIONS

Part B2 contains the service specifications.

B2.1 KMSA TO KMSA SERVICE SPECIFICATION

This section contains the KMSA-KMSA service specification.

B2.1.1 Available service primitives

The following service primitives are available:

KM-INIT;
NEW-KEY;
SECURITY-SERVICE;
KEY-UPDATE;
KM-RELEASE;
KM-ABORT.

B2.1.1.1 Key Management Initialization service primitive

The purpose of the Key Management Initialization (KM-INIT) service primitive is to establish a common basis between the two parties. The parties must agree about the algorithms to use. However, the algorithms to use may be preestablished and therefore the algorithm specifications parameters are declared as optional parameters.

Parameter	KM-INITreq	KM-INITind	KM-INITrsp	KM-INITcnf
Called Application Title	Mandatory	Mandatory (=)		
Calling Application Title	Mandatory	Mandatory (=)		
Resp Application Title			Mandatory	Mandatory (=)
Called Present Address	Mandatory	Mandatory (=)		
Calling Present Address	Mandatory	Mandatory (=)		
Resp Presentation Address			Mandatory	Mandatory (=)
Application Context Name	Optional	Optional	Optional	Optional
Confidentiality Specifications	Optional	Optional (=)		
Confidentiality Specification			Conditional	Conditional (=)
Integrity Specifications	Optional	Optional (=)		
Integrity Specification			Conditional	Conditional (=)
Key Exchange Specifications	Optional	Optional (=)		
Key Exchange Specification			Conditional	Conditional (=)
State Result			Mandatory	Mandatory
Diagnostic			Optional	Conditional

Called Application Title

The called application title is the title used to identify the key management entity acting as responder. The value is an application entity title.

Calling Application Title

The calling application title is the title used to identify the initiating key management entity. The value is an application entity title.

Responding Application Title

The responding application title is that title returned by the responder. The value is an application entity title.

Called Presentation Address

The called presentation address is the address used by the calling service to identify the PSAP to which the association is to be establish.

Calling Presentation Address

The calling presentation address is the address from which the association is established.

Responding Presentation Address

The responding presentation address is the address which is used to reestablish the association after failure.

Application Context Name

The application context name parameter carries a name to represent the properties of the association as a whole. The initiator proposes a name which may be accepted, and returned by the responder, or the responder may return a different name. In either case the name returned by the responder is the application context name applicable to the established association.

Confidentiality Specifications

This parameter is declared as optional because the two parties may have preestablished the confidentiality algorithm to use. If the parties have not made any prior agreements this parameter must be present. The initiator proposes a set of supported confidentiality algorithms. This parameter has to be used in consistency with the local security policy.

Confidentiality Specification

The parameter can only be present if parameter Confidentiality Specifications is present. The responder selects one algorithm within the set proposed by the initiator.

Integrity Specifications

This parameter is declared as optional because the two parties may have preestablished the integrity algorithm to use. If the parties have not made any prior agreements this parameter must be present. The initiator proposes a set of supported integrity algorithms. The parameter has to be used in consistency with the local security policy.

Integrity Specification

This parameter can only be present if Integrity Specifications is present. The responder selects one algorithm within the set proposed by the initiator.

Key Exchange Specifications

This parameter is declared as optional because the two parties may have preestablished the key exchange algorithm to use. If the parties have not made any prior agreements this parameter must be present. The initiator proposes a set of supported key exchange algorithms. The parameter has to be used in consistency with the local security policy.

Key Exchange Specification

This parameter can only be present if Key Exchange Specifications is present. The responder selects one algorithm within the set proposed by the initiator.

State Result

If State Result indicates failure the next state of the protocol machine is not reached. The parameter may be used to indicate that the responder cannot support the algorithm proposed by the initiator.

Diagnostic

The diagnostic parameter clarifies the reason to why the State Result parameter has a failure value.

B2.1.1.2 New Key service primitive

The purpose of this service primitive is to establish the traffic encryption keys. The parameters included in this service primitive depend on the selected key exchange algorithm.

Parameter	NEW-KEYreq	NEW-KEYind	NEW-KEYrsp	NEW-KEYcnf
Algorithm Id	Optional	Optional (=)	Conditional	Conditional (=)
Algorithm Dependent Data	Optional	Optional	Optional	Optional

Algorithm Id

Identifies the key exchange algorithm used. The value may be preestablished.

Algorithm Dependent Data

The parameter contains the key exchange data specific for a certain algorithm. The algorithm may be identified may by the parameter Algorithm Id. The parameter is declared optional because the data may be handled by the key management application entity.

B2.1.1.3 Security Service service primitive

The purpose of this service primitive is to negotiate about the attributes of a particular security protocol.

Parameter	SEC-SERVreq	SEC-SERVind	SEC-SERVrsp	SEC-SERVcnf
Said	Mandatory	Mandatory	Mandatory	Mandatory
Algorithm Id	Optional	Optional (=)	Conditional	Conditional(=)
Algorithm Dependent Data	Optional	Optional (=)	Optional	Optional (=)
Security Protocol Id	Mandatory	Mandatory (=)	Mandatory	Mandatory (=)
Security Protocol Attributes	Mandatory	Mandatory	Mandatory	Mandatory

Said (Security Association Identifier)

An identifier which makes it possible later on to recognize the established security association.

Algorithm Id

Identifies the key exchange algorithm used. The value may be preestablished.

Algorithm Dependent Data

The parameter contains data that is specific for a certain algorithm. The algorithm may be identified by the parameter Algorithm Id.

Security Protocol Id

Identifies a particular security protocol.

Security Protocol Attributes

Contains the attributes which are associated with a particular Security Protocol.

B2.1.1.4 Key Update service primitive

The purpose of this service primitive is to update the traffic encryption keys. The updated traffic encryption keys are included in the security association recognized by New Local Said.

Parameter	KEY-UPDreq	KEY-UPDind	KEY-UPDrsp	KEY-UPDcnf
Remote Said	Mandatory	Mandatory	Mandatory	Mandatory
New Local Said	Mandatory	Mandatory	Mandatory	Mandatory

Remote Said

Identifies the security association at the receiving side, that is, of the key management process receiving KEY-UPDind or KEY-UPDcnf.

New Local Said

The new local identifier for the key management process issuing KEY-UPDreq or KEY-UPDrsp.

B2.1.1.5 KM-Release service primitive

The KM-RELEASE service primitive is used to terminate the application association, it is the normal way to terminate the association. KM-Release has no parameters.

B2.1.1.6 KM-Abort service primitive

The purpose of this service primitive is to terminate the current application association.

Parameter	KM-ABORTreq	KM-ABORTind
State Result	Mandatory	Mandatory
Diagnostic	Optional	Optional (=)

State Result

Gives an indication that an error condition has occurred.

Diagnostic

The diagnostic parameter conveys detailed information about the failure of the requested action. The parameter has different meaning depending on in which state the error action did occur.

B2.1.2 Security protocol parameters

This section contains the specification of the attributes which are associated with a particular security protocol.

B2.1.2.1 TLSP parameters

When TLSP is selected as security protocol the data defined herein appears in the Security Protocol specification field of the Security Service primitive.

Parameter	SEC-SERVreq	SEC-SERVind	SEC-SERVrsp	SEC-SERVcnf
Confidentiality	Mandatory	Mandatory	Mandatory	Mandatory
Integrity	Mandatory	Mandatory	Mandatory	Mandatory
Security Levels	Optional	Optional (=)		
Security Labels	Optional	Optional(=)		
Security Level			Conditional	Conditional (=)
Security Label			Conditional	Conditional (=)
Kind of Key	Mandatory	Mandatory	Mandatory	Mandatory

Confidentiality

Specifies whether a confidentiality service will be included in the security association.

Integrity

Specifies whether an integrity service will be included in the security association.

Security Levels

Specifies the security levels associated with the key material. The initiator may propose a set of security level definitions.

Security Level

The responder selects one security level within the set proposed by the initiator.

Security Labels

The Security Labels parameter specifies the security label format that is going to indicate the security levels as specified by parameter Security Levels. The initiator may propose a set of supported security label syntaxes. Each Security Label definition matches a Security Level definition.

Security Label

The responder selects a security label representation within the set proposed by the initiator. The selected security label has to match the selected security level.

Kind of Key

Decides wether TLSP will be used on a per connection or on a per NSAP basis.

B2.2 KMSA TO KCA service primitives

This section contains the KMSA-KCA service specification.

B2.2.1 Available service primitives

The following service primitives are available:

KM-INIT;
NEW -KEY;
REKEY;
REKEY-DELIVERY;
KM-RELEASE;
KM-ABORT.

B2.2.1.1 Key Management Initialization service primitive

Two parameters have been added compared to the definitions of 2.1.1.1. These parameters are:

Rekey specifications

This parameter is declared optional because the two parties may have preestablished the rekey method to use. If the parties have not made any prior agreements this parameter must be present. The initiator proposes a set of supported rekey methods. The parameter has to be used in consistency with the local security policy.

Rekey specification

This parameter can only be present if the Rekey specifications parameter is present. The responder selects one method within the set proposed by the initiator.

B2.2.1.2 New Key service primitive

See B2.1.1.2.

B2.2.1.3 Rekey service primitive

Parameter	REKEYreq	REKEYind	REKEYrsp	REKEYcnf
Algorithm Id	Optional	Optional (=)	Conditional	Conditional (=)
Algorithm Dependent Data	Optional	Optional	Optional	Optional

Algorithm Id

Identifies the rekey method used. The parties may have preestablished the algorithm to use.

Algorithm Dependent Data

The parameter contains the rekey data specific for a particular algorithm. The algorithm may be identified by the parameter Algorithm id. The parameter is declared optional because the rekey data may be handled by the key management application entity.

B2.2.1.4 Rekey Delivery service primitive

Parameter	REKEY-DELreq	REKEY-DELind	REKEY-DELrsp	REKEY-DELcnf
Algorithm Id	Optional	Optional (=)	Conditional	Conditional(=)
Algorithm Dep. Data	Optional	Optional	Optional	Optional

Algorithm Id

Identifies the rekey method used. This value may be preestablished.

Algorithm Dependent Data

The parameter contains the rekey (delivery) data specific for a certain algorithm. The algorithm may be identified by the parameter Algorithm id. The parameter is declared optional because the rekey delivery data may be handled by the key management application entity.

B.2.2.1.5 KM-Release service primitive

See 2.1.1.5.

B.2.2.1.6 KM-Abort service primitive

See B.2.1.1.6.

B2.3 KCA TO KCA service primitives

Reserved for the KCA to KCA service specification.

B2.4 GENERIC AND ALGORITHM DEPENDENT PARAMETERS

This section contains the definitions of the parameters of generic or algorithm dependent nature.

B2.4.1 Key Exchange Methods

The data defined herein will appear in the Algorithm dependent parameter field of the New Key service primitive.

B2.4.1.1 Diffie-Hellman

Parameter	NEW-KEYreq	NEW-KEYind	NEW-KEYrsp	NEW-KEYcnf
Cert Path	Optional	Optional	Optional	Optional
Unidirectional Keys	Optional	Optional	Optional	Optional
D-H Publ. Key Material	Optional	Optional	Optional	Optional

Certification Path

Can contain a certification path. However, the certification path sent to the peer entity must be sent in its transfer syntax form. Therefore, in order to avoid unnecessary re-translations, the certificates may be cached and managed by the key management application entity. Therefore this parameter is optional.

D-H Public Key Material

Can contain the Diffie-Hellman public key. However, the key needs to be signed, the signature must be computed on the transfer syntax representation which normally is not available for a key management application process. An alternative is to let this data be available only for the key management application entity. Therefore the parameter is optional.

Unidirectional Keys

Tells whether the created symmetric keys are going to be unidirectional or not. If unidirectional keys are used the entities will have both a sending and receiving key.

DIAGNOSTICS

Here is a suggestion of some diagnostic parameters related to the New Key Request service primitive:

Type	Reason
0	Failed to authenticate
1	Argument not accepted
2	Certification path not complete
3	Do not support unidirectional keys

B2.4.1.2 RSA

Parameter	NEW-KEYreq	NEW-KEYind	NEW-KEYrsp	NEW-KEYcnf
Certification Path	Optional	Optional	Optional	Optional
Unidirectional	Optional	Optional	Optional	Optional
RandomA	Optional	Optional		
RandomB			Optional	Optional

Certification Path

See previous description.

RandomA

Can contain the random number which is encrypted with the responder´s public key. The encryption of the random number must be performed on the transfer syntax representation. Therefore, as an alternative the number may be available only for the key management application entity. How it is handled is locally dependent.

RandomB

Can contain the random number which is encrypted with the responder´s private key and initiator´s public key. The same problems as the above mentioned is encountered.

Unidirectional keys

See previous description.

B2.4.2 Rekey Methods

The data defined herein will appear in the Algorithm dependent parameter fields of the Rekey and Rekey-Delivery service primitives.

B2.4.2.1 rekMeth1

Parameter	REKEYreq	REKEYind	REKEYrsp	REKEYcnf
Public Key	Optional	Optional (=)		
Transaction			Optional	Optional (=)

Parameter	REKEY-DELreq	REKEY-DELind	REKEY-DELrsp	REKEY-DELcnf
Transaction	Optional	Optional(=)		
Certificate			Optional	Optional(=)

Public Key

The public key to be certified. The public key may be available only for the key management application entity. Therefore the parameter is optional.

Transaction

A number associated with the transaction.

Certificate

The newly created certificate. The certificate may be available only for the key management application entity. Therefore the parameter is optional.

B2.4.3 Diagnostics

Here are some suggestions on diagnostic information. It is up to the local security policy to decide whether it can be sent. The concern is that to much information would be provided to an intruder.

KMINIT related diagnostics

Type	Reason
0	Proposed Confidentiality Algorithms not supported
1	Proposed Integrity Algorithms not supported
2	Proposed Key Exchange Methods not supported

NEW KEY related diagnostics

The diagnostic parameters are defined in the algorithm dependent section.

SECURITY SERVICE related diagnostics

Type	Reason
0	PDU could not be correctly decrypted

There are algorithm dependent diagnostic parameters, these parameters begin with the value 100. There are also security protocol dependent diagnostic parameters, they begin with the value 200.

B2.5 ALTERNATIVE SERVICE SPECIFICATION

One problem encountered in the service specification was whether some specific parameters should be provided or not. For example, should the certification path be included in the New Key service primitive? The motive for not including the certification path is that it might be desirable to have the certification path cached in its transfer syntax form, thereby avoiding unnecessary re-translations. Encryption and decryption procedures are performed on the transfer syntax representation of the data, therefore the Diffie-Hellman public key provided in the New Key request cannot be sent signed.

The service primitives issued, as previously described, can been seen as issued from the part of the key management application process that resides outside the key management application entity. Where should the "intelligence" be built in? In the part of the application process that is outside the application entity or in the application entity? For example, should the authentication procedure be employed in the application process or application entity? The specifications of [IEEE P802.10C] are made under the assumption that the "intelligence" is provided by the key management application entity. Currently [IEEE P802.10C] provides the following service primitives:

- KM Associate;
- Get Security Association;
- Delete Security Association;
- Set Security Association;
- Update Security Association;
- KM Release;
- KM Error.

The key management application process by issuing the Get Security Association service primitive causes the creation of a security association. The application process primarily provides the application entity information about the intended security protocol and the security services to be included in the security association. The service primitive causes the two key management application entities to exchange the Nkrq and Nkrs PDUs and thereafter the Ssrq and Ssrs PDUs. Such things as certification paths, certificate revocation list handling and handling of key material (such as the Diffie-Hellman public key) are entirely handled by the application entity. By doing so the problems encountered in B2.4 are avoided.

ANNEX C SUGGESTED READING

This paper does not primarily deal with the methods and algorithms involved when a key exchange occurs. This annex lists a number of articles where different key exchange principles are presented.

In the article The First Ten Years Of Public Key Cryptography [DIFF88], Witfield Diffie describes the Diffie-Hellman algorithm. The author also gives his view on an asymmetric versus a symmetric form of key management.

In the paper Discussions Of Symmetrical (Secret) Key And Asymmetrical (Public Key) Cryptography [BARK91], Barker and Graff describe symmetric and asymmetric key management techniques. Barker and Graff chose [ANSI X9.17] and [ANSI X9.28] to describe the symmetric form of key management. Diffie-Hellman, RSA, and ElGamal are chosen as the algorithms to describe an asymmetric form of key management.

The National Physical Laboratory (NPL) report Issues In The Design Of A Key Distribution Centre [Center] [PRIC81], by W. L. Price and D. W. Davies, gives examples of key exchanges based on RSA (asymmetric) and DES (symmetric).

A classic article, Using Encryption For Authentication In Large Networks Of Computers [NEED78] by Needham and Schroeder, describes how two users can obtain their traffic encryption keys from a Key Distribution Center. Key exchanges based on both asymmetric and symmetric principles are described.

The article Timestamps In Key Distribution Protocol by Dorothy E Denning and Giovanni Maria Sacco [DENN81] describes how timestamps can be used when distributing keys.

ANNEX D ASN.1 EXTENSIONS

The existing ASN.1 data types described in ISO 8824, Abstract Syntax Notation One [ISO 8824], are used to make the protocol definitions that appear in Annex B. At the moment (November 91) there are ongoing activities within ISO and CCITT to enhance ASN.1. Those enhancements affect protocol designers using macros, ANY and ANY DEFINED BY. The ASN.1 extensions are described in the paper Tutorial On "Table Types And Function" [STEE90].

The Security Exchange ASE (SE ASE) [ISO/SC21 6096] [ISO/SC21 6097] [ISO/SC21 6098] makes use of these ASN.1 enhancements. The SE ASE is used to exchange security information objects between two entities. The exchange can be either simple or complex. In Figure D:1 a three way complex exchange is illustrated[1].

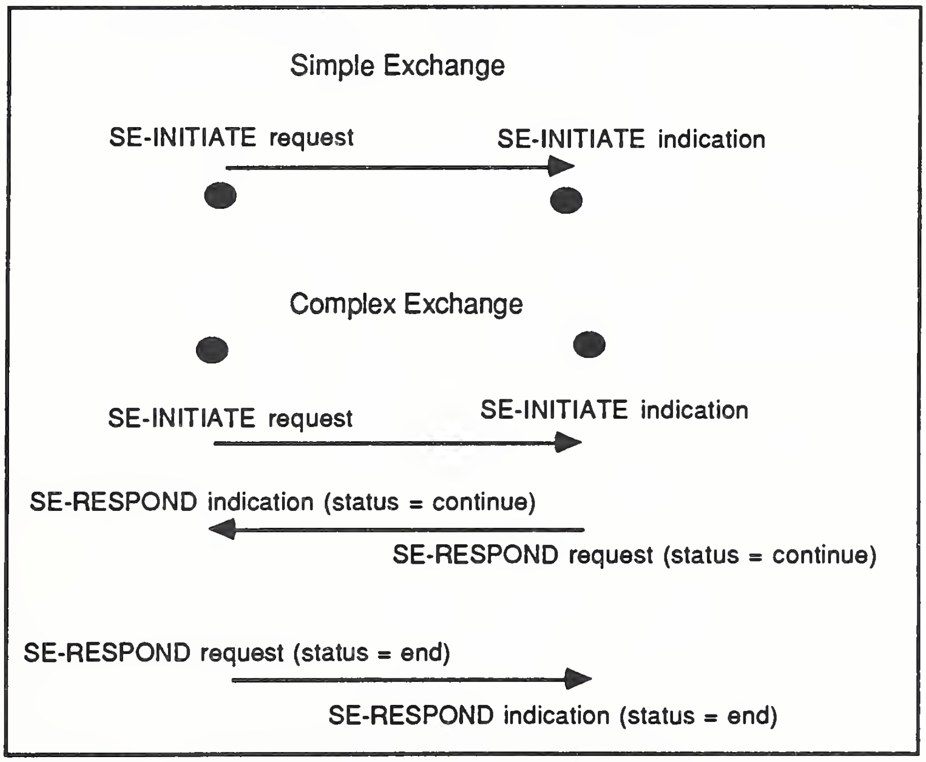

Simple Exchange
SE-INITIATE request
SE-INITIATE indication
Complex Exchange
SE-INITIATE request
SE-INITIATE indication
SE-RESPOND indication (status = continue)
SE-RESPOND request (status = continue)
SE-RESPOND request (status = end)
SE-RESPOND indication (status = end)

Figure D:1 Simple and Complex exchange

The Upper Layer Security Model [ISO/SC21 6095] defines a security information object (SIO) to be a logical piece of security exchange information which is transported from one system to another. A SIO-set is the set of SIOs transported as part of the sequence of transfers in a security exchange. The definition of the security exchange information for a particular security exchange involves providing (at the abstract syntax level) definitions of the data types corresponding to the respective SIOs in the SIO-set. There are ongoing activities in ISO SC 27 WG1 [ISO/SC27 169]

[1] Note, a three way exchange is only used for illustration purposes, there are no restrictions in [14], [15] and [16] about the number exchanges that can occur.

in the security information objects area. In the near future, exchanges of security information objects may be registered.

One way to view a key exchange is like a complex security exchange. In each exchange an SIO is transferred, and each SIO is part of a SIO-set. The security objects exchanged are registered in a security register.

This paper does not discuss the SE ASE very much, because the awareness that the SE ASE could be used for the exchange of keys came very late.

A possibility is to use the ASN.1 enhancements mentioned already when defining the layer seven key management protocol of Annex B. The drawbacks at this moment are:

- If the goal is to have the protocol specification ready in a short term, it may be hazardous to base it on features which are not yet standardized. Often there are a lot of changes made during a standardization process.
- ASN.1 parsers must be enhanced to be able to parse the ASN.1 extensions. It is not possible to implement the key management protocol without ASN.1 parsers.

GLOSSARY

OSI TERMS AND ABBREVIATIONS

In this section terms and abbreviations of general nature are defined. The terms and abbreviations are taken from [ROSE90a].

Abstract syntax: A description of a data structure that is independent of machine oriented structures and restrictions.

Abstract Syntax Notation One: The OSI language for describing abstract syntax.

ACSE: See Association Control Service Element.

American National Standards Institute: The U.S. national standardization body. The ANSI is a member of the ISO.

ANSI: See American National Standards Institute.

Application entity: The OSI portion of an application process (AP),

Application layer: That portion of an OSI system ultimately responsible for managing communication between application processes (APs).

Application process: An object executing in a real system.

Application service element: The building block of an application entity (AE). Each AE consists of one or more of these service elements, as defined by its application context.

ASN.1: See Abstract Syntax Notation One.

Association Control Service Element: The application service element responsible for association establishment and release.

CMIP: See Common Management Information Protocol.

CMISE: see Common Management Information Service Element.

CCITT: See International Telephone and Telegraph Consultative Committee.

Common Management Information Protocol: The OSI protocol for network management.

Common Management Information Service Element: The application service element responsible for carrying network management semantics.

Connection-less mode: A service that has a single phase involving control mechanisms such as addressing in addition to data transfer.

Connection-oriented mode: A service that has three distinct phases: *establishment*, in which two or more users are bound to a connection; *data transfer*, in which data is exchanged between the users; and, *release*, in which binding is terminated.

DAP: See Directory Access Protocol.

Directory Access Protocol: The protocol used between a Directory User Agent (DUA) and a Directory System Agent (DSA).

Directory Information Tree: The global tree of entries corresponding to information objects in the Directory.

Directory System Agent: An application entity that offers the Directory service.

Directory User Agent: An application entity that makes the Directory service available to the user.

Distinguished Name: The global, authoritative name of an entry in the OSI Directory.

Directory System Protocol: The protocol used between two Directory System Agents.

DIT: See Directory Information Tree.

DSA: See Directory System Agent.

DUA: See Directory User Agent.

ECMA: See European Computer Manufacturer Association.

European Computer Manufacturer Association: A group of computer vendors that have performed substantive prestandardization work for OSI.

File Transfer, Access and Management: The OSI file service.

FTAM: See File Transfer, Access and Management.

IEEE: See Institute of Electrical and Electronics Engineers.

Initiator: A service user that initiates a connection or association.

Institute of Electrical and Electronics Engineers: A profession organization, which, as part of its services to the community, performs some prestandardization work for OSI.

International Organization for Standardization: The organization that produces much of the world´s standards. OSI is only one of many areas standardized by ISO/IEC.

International Telephone and Telegraph Consultative Committee: A body comprising the national Post, Telephone, and Telegraph (PTT) administrations.

ISO/IEC: See International Organization for Standardization.

Network layer: That portion of an OSI system responsible for data transfer across the network, independent of both the media comprising the underlying subnetworks and the topology of those subnetworks.

NSAP: Network service access point.

Open Systems Interconnection: An international effort to facilitate communications among computers of different manufacture and technology.

OSI: See Open Systems Interconnection.

Presentation layer: That portion of an OSI system responsible for adding structure to the units of data that are exchanged.

Protocol control information: (Conceptually) the first part of a protocol data unit used by a protocol machine to communicate information to its peer.

PDU: See protocol data unit.

Protocol data unit: A data object exchanged by protocol machines, usually containing both protocol control information and user data.

Protocol machine: A finite state machine (FSM) that implements a particular protocol. When a particular input (e.g., service request or network activity) occurs in a particular state, the FSM potentially generates a particular output (e.g., service indication or network activity) and possibly moves to another state.

Relative Distinguished Name: A component of an entry´s Distinguished Name, usually consisting of an attribute/value pair.

Request for Comments: The documents series describing the Internet suite of protocols and related experiments.

Responder: A service user that responds to a connection or association.

RFC: See Request for Comments.

QOS: See Quality of Service.

Quality of Service: The desired or actual characteristics of a service; typically, but not always, those of the network service.

SAP: See service access point.

SDU: See service data unit.

Service access point: An artifact modeling how a service is made available to a user.

Service data unit: User-data passed through a service access point.

Service primitive: An artifact modeling how a service is requested or accepted by a user.

Session layer: That portion of an OSI system responsible for adding control mechanisms to the data exchange.

Transfer syntax: A description of an instance of a data structure that is expressed as string of bits.

Transport layer: That portion of an OSI system responsible for reliability and multiplexing of data across network (over and above that provided by the network layer) to the level required by the application.

TSAP: Transport service access point.

SECURITY GLOSSARY

In this section security terms and abbreviations used in this paper are defined. The terms and abbreviations are compiled mainly from [ISO 7498-2], [ISO/SC21 6693] and [BARK91].

Access Control: The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner.

Asymmetric cryptographic algorithm: A cryptographic algorithm that uses two related keys. At least one of the two keys is the cryptographic inverse of the other.

CA: Certification Authority.

Confidentiality: The property that information is not made available or disclosed to unauthorized individuals, entities, or processes.

Connection confidentiality: This service provides for the protection of (N)-user-data on an (N)-connection.

Connectionless confidentiality: This service provides for the confidentiality of all (N)-user-data in a single connectionless (N)-SDU.

Connection Integrity with recovery: This service provides for the integrity of all (N)-user-data on an (N)-connection and detects any modification, insertion, deletion or replay of any data within an entire SDU sequence (with recovery attempted).

Connection Integrity without recovery: As above but with no recovery attempted.

Cryptoanalysis: The analysis of a cryptographic system and/or its input and outputs to derive confidential variables and/or sensitive data including cleartext.

Cryptography: The discipline which embodies principles, means, and methods for the transformation of data in order to hide its information content, prevent its undetected modification and/or prevent unauthorized use.

Data integrity: The property that data has not been altered or destroyed in an unauthorized manner.

Data origin authentication: The corroboration that the source of data received is as claimed.

Decryption[1] : The reversal of a corresponding reversible encipherment.

DES: Data Encryption Standard.

Digital signature: Data appended to, or a cryptographic transformation (see cryptography) of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery e.g. by the recipient.

Encryption[2]: The cryptographic transformation of data (see cryptography) to produce ciphertext.

Exclusive-OR: A mathematical operation (symbol $\oplus$) defined as:

$$0 \oplus 0 = 0$$
$$1 \oplus 0 = 1$$
$$0 \oplus 1 = 1$$
$$1 \oplus 1 = 0$$

Exponentiation: A mathematical process where one number is raised to some power. For example, 2 raised to the third power is written $2^3 = 2 \times 2 \times 2 = 8$

Hash function: An algorithm which processes a variably-sized message as input and produces a fixed-size representation (i.e., a message digest) of that message as output.

Key: A sequence of symbols that controls the operations of encipherment and decipherment.

Masquerade: The pretence by an entity to be a different entity.

---

[1] Decipherment is an equivalent term.

[2] Encipherment is an equivalent term.

Peer-entity authentication: The corroboration that a peer entity in an association is the one claimed.

Plaintext: Unencrypted (unenciphered) data.

Private key: The key, used in an asymmetric algorithm, that is known to only one entity.

Public key: The key, used in an asymmetric algorithm, that is publicly available.

RSA: Rivest, Shamir, Adleman (public key encryption algorithm).

Security domain: A set of elements, a security policy, an authority and a set of relevant activities in which the set of elements are subject to the security policy, administered by the authority, for the specified activities.

Secret key: In a symmetric encipherment algorithm the key shared between two entities.

Security label: The marking bound to a resource (which may be a data unit) that names or designates the security attributes of that resource.

Security policy: The set of criterias for the provision of security services.

Security service: A service, provided by a layer of communicating open systems, which ensures adequate security of the systems or of data transfers.

Security Management Information Base: A conceptual repository for all security information needed by open systems.

SMIB: Security Management Information Base.

Symmetric cryptographic algorithm: A cryptographic algorithm that uses a single key for both encryption and decryption.

Traffic analysis: The inference of information from observation of traffic flows (presence, absence, amount, direction and frequency).

Traffic flow confidentiality: A confidentiality service to protect against traffic analysis.

Trust: A relationship between two elements, a set of activities and a security policy in which element x trusts element y if and only if x has confidence that y will behave in a well defined way (with respect to the activities) that does not violate the given security policy.

ANSI X9 SECURITY TERMS AND ABBREVIATIONS

In this section the terms and abbreviations used when describing [ANSI X9.17] and [ANSI X9.28] are defined. The reason why these terms are not mixed with the previously defined security terms, is that some of the terms defined in [ANSI X9.17] and [ANSI X9.28] are in conflict with correspondent terms in [ISO 7498-2]. This is particularly valid for terms concerning authentication.

The following terms and abbreviations are defined in [ANSI X9.17] and [ANSI X9.28].

Authentication: The act of determining that a message has not been changed since leaving its point of origin. The identity of the originator is implicity verified.

Cryptographic Service Message: A message for transporting keys or related information used to control a keying relationship.

Error Service Message: Used to give a negative acknowledgment on receipt of any Cryptographic Service Message other than an ESM and to give the recipient data with which to recover.

ESM: See Error Service Message.

Initialization Vector (IV): A number used as a starting point for encryption of a data sequence to increase security by introducing additional cryptographic variance and to synchronize cryptographic equipment.

Key Encrypting Key: A key used exclusively to encrypt and decrypt keys.

Key exchange transaction: A set of CSM messages used to transport a set of subscriber keys within a multiple center group.

Key offset: The process of exclusive-or´ing a counter to a key.

Keying relationship: The state existing between a communicating pair during which time they share at least one data key or key encrypting key.

Key Service Message: Used to transfer keys between communicating pair.

KSM: See Key Service Message.

Message Authentication Code: A number which is the result of passing a message through the authentication algorithm using a specific key.

MRFS: See Multiple center Request For Service.

MRSI: See Multiple center Request Service Initiation.

MRSM: See Multiple center Response Service Message.

MRTR: See Multiple center Response To Request.

Multiple center Request For Service: Similar to RFS.

Multiple center Request Service Initiation. Similar to RSI.

Multiple center Response Service Message. Similar to RSM.

Multiple center Response To Request: Similar to RTR.

Notarization: A method of applying additional security to a key utilizing the identities of the originator and the ultimate recipient.

Offset: See Key Offset.

Party [in the X9.17 and X9.28 standards it is defined as logical party]: One or more physical parties that form one member of a communicating pair.

Request Service Initiation: Optionally used to request key from another party.

Response Service Message: Used to provide an authenticated acknowledgment.

Response To Request Message: Used to send keys from a Key Distribution Center or from a Key Translation Center.

RSI: See Request Service Initiation.

RSM: See Response Service Message.

RTR: See Response To Request Message.

Subscriber: An entity obtaining multiple center key management service from a multiple center group via a multiple center agent.

REFERENCES

[ANSI X9.9]	ANSI X9.9, Financial Institution Message Authentication.
[ANSI X9.17]	ANSI X9.17, Financial Institution Key Management (Wholesale), April 4, 1985.
[ANSI X9.28]	ANSI X9.28, Financial Institution Key Management, July 5, 1990.
[DoD 5200.28]	National Computer Security Center, Department of Defense Trusted Computer Security Evaluation Criteria, DOD 5200.28-STD.
[ECMA TR/46]	Standard ECMA TR/46, Security in Opens Systems: A Security Framework.
[ECMA-138]	Standard ECMA-138, Security in Open Systems: Data Elements and Service Definitions.
[FIPS 46-1]	FIPS 46-1, Data Encryption Standard, National Institute of Standards and Technology.
[FIPS 171	Draft FIPS, Key Management Using ANSI X9.17, National Institute of Standards and Technology.
[FIPS 140-1]	Draft FIPS 140-1, Security Requirements For Cryptographic Modules, National Institute of Standards and Technology.
[IEEE P802.10B]	IEEE P802.10B/D6, Standard for Interoperable Local Area Networks (LAN) Security (SILS), Part B -- Security Data Exchange, November 5, 1990.
[IEEE P802.10C]	IEEE Standard for Interoperable Local Area Networks (LAN) Security, Working Draft, Part C -- Key Management Proposal, January 27, 1992.
[ISO 7498]	ISO 7498, Informations Processing Systems - Open Systems Interconnection, Basic Reference Model.
[ISO 7498-2]	ISO 7498-2, Informations Processing Systems - Open Systems Interconnection, Basic Reference Model, Security Architecture.
[ISO 8072]	ISO 8072, Information processing systems - Open Systems Interconnection - Connection oriented transport service definition.
[ISO 8073]	ISO 8073, Information processing systems - Open Systems Interconnection - Connection oriented transport protocol specification.
[ISO 8571-2]	ISO 8571, Information Processing Systems - Open Systems Interconnection - File Transfer, Access and Management, Part 2: Virtual Filestore Definition.

[ISO 8571-4]	ISO 8571, Information Processing Systems - Open Systems Interconnection - File Transfer, Access and Management, Part 4: File Protocol Specification.
[ISO 8649]	ISO 8649, Information Processing Systems - Open Systems Interconnection - Service Definition of the Association Control Service Element.
[ISO 8649/AD1]	ISO 8649, Information Processing Systems - Open Systems Interconnection - Service Definition of the Association Control Service element, Addenduml: Peer entity authentication during association establishments.
[ISO 8650]	ISO 8650, Information Processing Systems - Open Systems Interconnection - Protocol Specification for the Association Control Service Element.
[ISO 8650/AD1]	ISO 8650, Information Processing Systems - Open Systems Interconnection - Protocol Specification for the Association Control Service Element, Addenduml: Peer entity authentication during association establishments.
[ISO 8732]	ISO 8732, Banking - Key management (wholesale).
[ISO 8824]	ISO 8824, Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).
[ISO 8825]	ISO 8825, Information Processing Systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).
[ISO 9545]	ISO/IEC 9545, Information Processing Systems - Open Systems Interconnection - Application Layer Structure.
[ISO 9545/P1]	ISO/IEC 9545/PDAM1, Information Processing Systems - Open Systems Interconnection - Application Layer Structure, Proposed Draft Amendment.
[ISO 9072-1]	ISO 9072, Information Processing Systems - Text Communication - Remote Operations Part 1: Model, Notation and Service Definition, International Organization for Standardization and International Electrotechnical Committee.
[ISO 9072-2]	ISO 9072, Information Processing Systems - Text Communication - Remote Operations Part 2: Protocol Specification, International Organization for Standardization and International Electrotechnical Committee.
[ISO 9594-1]	ISO 9594, Information Technology - Open Systems Interconnection - The Directory - Part 1: Overview of concepts, models and services.
[ISO 9594-2]	ISO 9594, Information Technology - Open Systems Interconnection - The Directory - Part 2: Models.

[ISO 9594-3]	ISO 9594, Information Technology - Open Systems Interconnection - The Directory - Part 3: Abstract service definitions.
[ISO 9594-4]	ISO 9594, Information Technology - Open Systems Interconnection - The Directory - Part 4: Procedures for distributed operation.
[ISO 9594-5]	ISO 9594, Information Technology - Open Systems Interconnection - The Directory - Part 5: Protocol specifications.
[ISO 9594-6]	ISO 9594, Information Technology - Open Systems Interconnection - The Directory - Part 6: Selected attribute types.
[ISO 9594-7]	ISO 9594, Information Technology - Open Systems Interconnection - The Directory - Part 7: Selected object classes.
[ISO 9594-8]	ISO 9594, Information Technology - Open Systems Interconnection - The Directory - Part 8: Authentication Framework.
[ISO 9595]	ISO 9595, International Processing Systems, Open Systems Interconnection - Common Management Information Service Definition.
[ISO 9596]	ISO 9596, International Processing Systems, Open Systems Interconnection - Common Management Information Protocol Specification.
[ISO 9979]	ISO/IEC/DIS 9979.2, Data cryptographic techniques - Procedures for the registration of cryptographic algorithms, January 5, 1990.
[ISO 10164]	ISO/IEC CD 10164-9.2, Information Technology - Open Systems Interconnection - Systems Management - Part 9: Objects and Attributes for Access Control.
[ISO 10181-2]	ISO DIS 10181-2, Information Technology - Security Frameworks for Open Systems - Part 2: Authentication Framework.
[ISO 10181-3]	ISO/CD 10181-3, Information Technology - Security Frameworks for Open Systems - Part 3: Access Control Framework.
[ISO 10736]	ISO/IEC DIS 10736, Information Technology - Telecommunications and Information Exchange Between Systems - Transport Layer Security Protocol, October 11, 1991.
[ISO 10736/P]	ISO/IEC DIS 10736 PDAM1 - Information Technology - Telecommunications and Information Exchange Between Systems - Transport Layer Security Protocol. Amendment I: Security Association Establishment Protocol.
[ISO 11166]	ISO/CD 11166, Banking - Key Management by means of Asymmetric Algorithms, March 3, 1991.
[ISO 11166-2]	ISO/CD 11166-2, Banking - Key management by means of asymmetric algorithms - Part 2 - Approved algorithms using the RSA cryptosystem, March 12, 1991.

[ISO 11577]	ISO/IEC CD 11577, Information Technology, Telecommunications and Information Exchange Between Systems - Network Layer Security Protocol, November 13, 1991.
[ISO/SC21 6096]	ISO/IEC JTC1/SC 21 N 6096, Working Draft Security Exchange ASE - Part 1: Security Exchange, Model and Specification Framework.
[ISO/SC21 6097]	ISO/IEC JTC1/SC 21 N 6097, Working Draft Security Exchange ASE - Part 2: Service Definition.
[ISO/SC21 6098]	ISO/IEC JTC1/SC 21 N 6098, Working Draft Security Exchange ASE - Part 3: Protocol Specification.
[ISO/SC21 6167]	ISO/IEC JTC1/SC 21 N 6167, Working Draft Guide to Open Systems Security.
[ISO/SC6 6957]	ISO/IEC JTC1/SC6 6957, Working Draft Lower Layer Security Guidelines.
[ISO/SC21 6693]	ISO/IEC JTC/SC 21/6693, International Organisation for Standardization, Working Draft Security Framework Overview.
[ISO/SC21 6095]	ISO/IEC JTC/SC 21 N6095, International Organisation for Standardization, Working Draft Upper Layer Security Model.
[ISO/SC27 169]	ISO/IEC JTC1/SC27/WG1 N169, Working Draft Security Information Objects.
[NIST SEC REG92]	General Procedures for Registering Computer Security Objects, National Institute of Standards and Technology, January 14, 1992.
[RFC 1114]	RFC 1114, Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management, June 28, 1991.
[SDNS 90-4259]	Secure Data Network System (SDNS), Access Control Documents, NISTIR 90-4259.
[SDNS 90-4262]	Secure Data Network System (SDNS), Key Management Documents, NISTIR 90-4262.
[BACK88]	Backes, F., "Transporting Bridges for Interconnection of IEEE 802 LANs," IEEE Network, Volume 2, Number 1, January 1988.
[BARK91]	Barker, E. and Graff, J., "Discussions of Symmetrical (Secret Key) and Asymmetrical (Public Key) Cryptography," Internal NIST document, September 1991.
[DAVI89]	Davies, D. W. and Price, W. L., Security for Computer Networks, 2d. ed., John Wiley & Sons, 1989, Chapter 6, "Key Management,", pp. 133-168. Copyright © 1984, 1989 by John Wiley & Sons Ltd. Reprinted by permission of John Wiley & Sons, Ltd.

[DENN81]	Denning, D. E. and Giovanni, M. S., "Timestamps in Key Distribution Protocols," Communication of the ACM, Volume 24, Number 8, 1981, pp. 533-536.
[DIFF88]	Diffie, W., "The First Ten Years of Public Key Cryptography," Proceedings of the IEEE, Volume 76, Number 5, May 1988, pp. 560-577.
[DSSD87]	U.S. Congress, Office of Technology Assessment, Defending Secrets, Sharing Data: New locks and Keys for Electronic Information, OTA-CIT-310, Washington D.C., U.S. Government Printing Office, October 1987, Chapter 4, Security Safeguards and Practices, pp. 51-91.
[FUMY90]	Fumy, W. and Landrock, P., "Principles of Key Management," Proceedings of Symposium On Computer Security, Threats and Countermeasures, Rome, Italy November 1990, pp. 122-132 (Organized by Fondazione Ugo Bordoni).
[GASS89]	Gasser, M., Goldstein, A., Kaufman, C. and Lampson, B., "The Digital Distributed System Architecture," Proceedings of the 12th National Security Conference, Baltimore, Maryland, U.S.A., October 1989, pp. 305-319.
[GASS90]	Gasser, M., "The Role Of Naming in Secure Distributed Systems," Proceedings of Symposium on Computer Security, Threats and Countermeasures, Rome, Italy, November 1990, pp. 97-109 (Organized by Fondazione Ugo Bordoni).
[GORD 84]	Gordon, J., "Strong RSA Keys," Electronics Letters, Volume 20, Number 5, 1984, pp. 514-516.
[GRAFF90]	Graff, J., "Key Management Systems Combining X9.17 and Public Key Techniques," Proceedings of the 13th National Security Conference, Washington, D.C., U.S.A., October, 1990, pp. 49-61.
[HENS88]	Henshall, J. and Shaw, S., OSI Explained: End-To-End Communication Standard, Cichester, West Sussex, England, Ellis Horwood Limited, 1988, Chapter 5, "The presentation layer," pp. 59-73.
[HOUS88]	Housley, R., "Encapsulation Security Protocol For Local Area Networks," Local Area Network Security, eds. Berson, T. A. and Beth, T., Springer-Verlag, 1989, pp. 103-109.
[LAMB88]	Lambert, P., "Architectural Model of the SDNS Key Management Protocol," Proceedings of the 11th National Security Conference, Baltimore, Maryland, U.S.A., 1988, pp. 126-128.
[LAMB88b]	Handout from presentation of Lambert, P., OSI Implementors Workshop, Security Special Interest Group, Gaithersburg, Maryland, U.S.A., December 14-15, 1988.
[LINN89]	Linn, J. and Kent, S., "Privacy for Darpa Internet Mail," Proceedings of the 12th National Security Conference, Baltimore, Maryland, U.S.A., October 1989, pp. 215-229.

[LINN90]	Linn, J., "Practical Authentication for Distributed Computing," Proceedings of the 1990 IEEE Symposium on Security and Privacy.
[MATY78]	Matyas S. M. and Meyer C. M., "Generation, Distribution, and Installation of Cryptographic Keys," IBM System Journal, Volume 17, Number 2, 1978, pp. 126-137.
[MATY86]	Matyas S., "Initialization of Cryptographic Variables In a Network With a Large Number of Terminals," IBM Report TR 21.1000, August 1986.
[MEYE82]	Meyer, C. and Matyas, S., Cryptography: A New Dimension In Computer Security, John Wesley & Son, 1982, Chapter 4, "Communication Security and File Security Using Cryptography," pp. 192-270.
[MEYE82a]	Meyer, C. and Matyas, S., Cryptography: A New Dimension In Computer Security, John Wesley & Son, 1982, Appendix D, "Some Cryptographic Concepts and Methods of Attack," pp. 679-712.
[NECH91]	Nechtvatal, J., "Public Key Cryptography," NIST Special Publication 800-2, April 1991.
[NEED78]	Needham, R. M. and Schroeder, M. D., "Using Encryption for Authentication in Large Networks of Computers," Communications of the ACM, Volume 21, Number 12, 1978, pp. 93-99.
[NELS87]	Nelson, R., "SDNS Services And Architecture," Proceedings of the 10th National Security Conference, Baltimore, Maryland, U.S.A., September 1987, pp. 153-157.
[OMUR89]	Omura, J., Discussions On Draft Proposal For Public-Key Options For Key Management, Papers submitted to IEEE 802.10 Working Group, February 1989.
[PRIC81]	Price, W. L. and Davies, D. W., "Issues in The Design of A Key Distribution Centre," NPL Report DNACS 43/81.
[RAMA90]	Ramaswamy, R., "A Key Management Algorithm for Secure Communications in Open Systems Interconnection Architecture," Computer & Security, Number 9, 1990, pp. 77-84.
[ROSE90]	Rose, M., The Open Book: A Practical Perspective On OSI, Prentice-Hall, Englewood Cliffs, New Jersey, U.S.A., 1990.
[ROSE90a]	Rose, M., The Open Book: A Practical Perspective On OSI, Prentice-Hall, Englewood Cliffs, New Jersey, U.S.A., 1990, "Glossary," pp. 605-619. Editorial/production supervision: Jacqueline A. Jeglinski. Copyright © 1990 by Prentice-Hall Inc. Reprinted with permission by Prentice Hall.
[SEC LABEL90]	Security Labels For Open Systems An Invitational Workshop, Gaithersburg, Maryland, U.S.A., June 1990, NISTIR 4632.

[SEC LABEL91] Standard Security Labels For GOSIP An Invitational Workshop, Gaithersburg, Maryland, U.S.A., June 1991, NISTIR 4614.

[SMET91] Smetaniuk, B., "Distributed operation of the X.500 directory," Computer Networks and ISDN Systems, Number 21, 1991, pp. 17-40.

[STALL 90] Stalling, W., Handbook of Computer Communications Standards: Local Area Networks Standards, Howard W. Sams & Company, Carmel, Indiana, U.S.A., Chapter 8, "MAC Bridges," pp. 207-234.

[STEE90] Steedman, D., Tutorial On "Table Types And Functions", Paper submitted at CCITT meeting in Stockholm, Sweden, May, 1990.

[STEI88] Steiner, J. G, Neumann, C. and Schiller, J. I., "Kerberos: An Authentication Service for Open Network Systems," Version 4, Project Athena, Massachusetts Institute of Technology, Presented at the USENIX conference 1988, Dallas, Texas, U.S.A.

[TATE87] Tater, G. L. and Kerut, E. G., "The Security Data Networks System: An Overview," Proceedings of the 10th National Security Conference, Baltimore, Maryland, U.S.A., September 1987, pp. 150-152.

[VOYD84] Voydock, V. L. and Kent, S., "Security Mechanisms in a Transport Layer Protocol," Computer Networks, Number 8, 1984, pp. 433-449.

[WALK89] Walker, S. T., "Network Security: The Parts Of The Sum," Proceedings of the 1989 IEEE Symposium on Security and Privacy, pp. 2-9.

[x/OPEN TI] x/Open Portability Guide, Network Services: Transport Interface, Prentice-Hall, Englewood Cliffs, New Jersey, U.S.A., August 1988.

[ZANO90] Zanon, F., "An Analysis of the ISO 8732 - Key Management Standard," Proceedings of IFIP TC-11 6th International Conference, on Information Security, Helsinki, Finland, May 1990, pp. 95-114.