NAT'L INST. OF STAND & TECH R.I.C.
A11103 737711

NISTIR 4792

NIST
PUBLICATIONS

A Formal Description of the SDNS Security Protocol at Layer 4 (SP4)

Wayne A. Jansen

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

U.S. DEPARTMENT OF COMMERCE
Barbara Hackman Franklin, Secretary

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

NIST

QC
100
.U56
4792
1992
C.2

NISTIR 4792

A Formal Description of the SDNS Security Protocol at Layer 4 (SP4)

Wayne A. Jansen

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

March 1992

U.S. DEPARTMENT OF COMMERCE
Barbara Hackman Franklin, Secretary

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

ABSTRACT

The Secure Data Network System (SDNS) project, initiated by the National Security Agency in 1986, produced a computer network security architecture within the framework of the International Organization for Standardization (ISO) reference model for Open Systems Interconnection (OSI). This report contains a formal description of the SDNS security protocol at layer four (SP4), one component of the overall security architecture. Estelle is the OSI formal description technique (FDT) used for the SP4 specification. Estelle is based on an extended state transition model with language elements from the Pascal language. An Estelle specification describes a hierarchically structured system of modules. The design of the formal description is explained through a top level and subsequent level of module decomposition. A description of the underlying security management information base is also included.

Key Words: Secure Data Network System; Computer Network Security; Open Systems Interconnection; Security Protocol; Formal Description Technique

1. INTRODUCTION

1.1 Background

The National Institute of Standards and Technology (NIST) and the National Security Agency (NSA) initiated a joint project in Computer Network Security in 1984. The project was based on the recognition that a comprehensive set of security mechanisms is needed to provide cost effective access control to data in geographically distributed computer networks. While the detailed security mechanisms can differ between the classified and unclassified communities requiring security, both communities benefit when commercial computer networks are developed with a common security architecture. The NSA initiated the Secure Data Network System (SDNS) program in 1986 as a result, at least partially, of this joint project.

Security Protocol 4 (SP4) [ref. 3,4] is one element of the SDNS architecture [ref. 5], used to provide security services at the Transport Layer of the International Organization for Standardization (ISO) Basic Reference Model (BRM) for Open System Interconnection (OSI) [ref. 1]. SP4 is compliant with the security addendum to the OSI BRM [ref. 2], and forms the basis of the emerging ISO standard for a Transport Layer Security Protocol [ref. 6]. SP4 extends the OSI Transport connectionless and connection oriented services to provide or support the following security services defined in the security addendum:

(1) Data Integrity,

(2) Data Confidentiality,

(3) Data Origination Authentication, and

(4) Access Control.

SP4 functionality is logically situated at the bottom of the Transport Layer. It consists of a simple encapsulation/decapsulation protocol that protects normal Transport Protocol data units within a cryptographically secure envelope. SP4 is appropriate for security domains that are end-system oriented in their administration of security mechanisms such as encipherment, and require accountability and protection of data on a Transport connection basis. SP4 may be especially relevant to agencies and organizations who have determined that the required means of providing reliable end-to-end communications is through the class 4 Transport Protocol (e.g., the United States Government OSI Profile [ref. 12]). This is due to the tendency to administer such systems from a Transport standpoint, and the likelihood of extending this perspective to incorporate security services.

This document presents the design specifications for the formal description of SP4, as the initial step in the production of a reference implementation. The rationale for producing an SP4 reference implementation is to promote general understanding and acceptance of the standard, to remove errors and inconsistencies that may be present in its definition, and to encourage development of commercial product implementations. A reference implementation may be manually or automatically derived from the formal specification.

1.2 Approach

Estelle [ref. 7] is the formal description technique selected to specify the behavior of SP4 and model its operating environment. Compared with other formal description techniques, Estelle is more implementation oriented. Various tools have been developed at NIST, including the NIST Estelle specification compiler [ref. 8] and the Prototyping Environment for Distributed Systems (PEDS) preprocessor and runtime library [ref. 9]. They may be used to produce an implementation from an Estelle formal description, and provide a test and demonstration vehicle within the modeled environment. Key portions of the implementation can be used to produce a prototype implementation by replacing modeled components such as the communications network, by actual components within an end-system. The resulting prototype constitutes the reference implementation.

The design for the SP4 formal description concentrates on the SP4E subset of security services. SP4E provides security services for individual connections between end-systems on a collective basis, and has the advantage that it may be modeled as an independent protocol entity. The SP4E subset may be contrasted to the full functionality of SP4C that can provide security services individually to end-system connections but is dependent on mechanisms within the Transport Layer to provide those services. Although the full SP4C functionality is formally described, its use is constrained to the SP4E subset to avoid having to include a Transport Layer Protocol within the formal description. Therefore, the design and formal description given in this document emphasizes the SP4E subset.

The remainder of the document discusses the structure of the SP4E specification in terms of its top level modules (chapter 2), and reviews implications of the design choices made (chapter 3). Further decomposition of the top level modules is presented for the SP4 Machine (chapter 4) and its associated Key Manager (chapter 5). The organization of the underlying Security Management Information Base (chapter 6) is presented in terms of component objects and relationships.

The Estelle specification representing the top level design is provided in Appendix A. Appendices B and C contain further decompositions of top level modules. The detailed descriptions of attributes of modeled objects in the Security Management Information Base appear in Appendix D. The reader is assumed to be familiar with the SP4 standard and have a reading knowledge of the Pascal programming language.

2. SPECIFICATION FRAMEWORK

Estelle can be regarded as Pascal extended with constructs for specifying communications protocol descriptions, such as those associated with the representation and operation of finite state machines. The module is the principal means to refine or decompose a protocol description in Estelle. An Estelle specification describes a hierarchically structured system of modules. Modules are normally used to model a class of system process, and can be instantiated and coupled through available language constructs. Functions and procedures are supported as alternative means of functional decomposition and are typically used to hide implementation details. Asynchronous exchanges between modules are modelled by the interaction point construct. Interaction points are divided into roles (e.g., user, provider) that delineate the responsibilities of exchanged services within a dialogue.

The selection and organization of modules for the specification was a compromise among many conflicting objectives. The following objectives were key considerations:

(1) to illustrate SP4 in as clear and understandable a fashion as possible, maintaining established software engineering principles;

(2) to express the full generality and implications of the SP4E subset, in an implementation independent manner;

(3) to isolate and limit the modules that require a high level of trust in a secure application;

(4) to make the specification as realistic as possible, while avoiding placing limitations on future enhancements; and

(5) to keep the specification aligned with the SDNS architectural principles; in particular:

   (a) to rely on key management to provide authentication and access control for SP4 entities,

   (b) to allow different key management schemes and allied protocols to be used, and

   (c) to maintain the independence of security services from any underlying cryptographic security mechanisms.

At the highest module level, the SP4E specification contains common facilities shared by modules formed by its refinement. These facilities include a Cryptographic Facility where the cryptographic security mechanisms reside, and a Man-Machine Interface

Facility that contains machine dependent mechanisms supporting user interaction. Both facilities are represented as sets of primitive routines and therefore only their interface description is considered part of the formal description.

Refinement of the SP4E specification module produces the top level design, consisting of the following Estelle modules:

(1) the Transport User,

(2) the SP4E Entity,

(3) the Key Manager, and

(4) the Communications Network.

Figure 1 illustrates these modules in a typical test configuration, along with the group of functions and procedures that collectively make up the Cryptographic Facility (Cryptofacility).

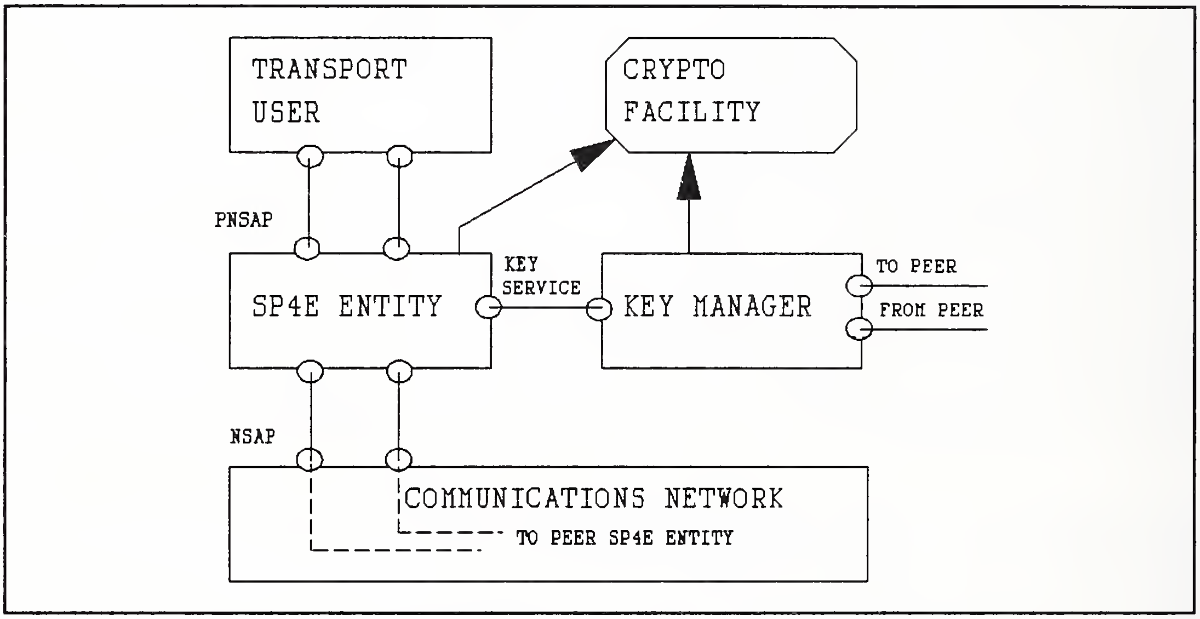

Figure 1: SP4E Simulated Environment

The SP4E Entity module is the cornerstone of the specification and forms the basis for the reference implementation. The remaining modules are included in the specification to provide a context and/or test harness for the SP4E Entity module. The SP4E Entity uses the Key Manager and the Cryptographic Facility shared with the Key Manager to provide security services for the Transport User over the Communications Network. The sections that follow describe each module in further detail. Appendix A gives the corresponding Estelle specification.

2.1 The Transport User

The Transport User module simulates one or more Transport Protocol entities. The module provides a demonstration capability that allows the selection and processing of test scenarios through the Man-Machine Interface Facility. It may be initialized to either an initiator or responder role corresponding to the desired handling and treatment of test scenario exchanges. The Transport User module employs the pseudo connectionless Network interface provided by the SP4E Entity to drive the simulated environment. The pseudo connectionless Network interface is an artifact provided by the SP4E Entity. The interface is an exact duplicate of that offered by the Communications Network, which it, in turn, employs.

2.2 The SP4E Entity

SP4E Entity modules communicate with one another through a connectionless Network interface offered by the Communications Network. The SP4E Entity module also provides a pseudo connectionless Network interface to the Transport User as mentioned previously. For each pseudo Network service access point (PNSAP) offered, there is an identical Network service access point (NSAP) utilized and vice versa. The pairing of service access points in this manner allows the SP4E security mechanisms to be isolated from both the Transport and Network Protocol entities, within a module. The SP4E Entity module also employs an asynchronous interface to the Key Manager. The PNSAP, the NSAP, and the interface to the Key Manager are modeled in Estelle as interaction points.

The SP4E Entity is composed of two types of submodules: the SP4 Machine and the Timer Facility. The SP4 Machine submodule encapsulates and decapsulates Transport Protocol data units within a security envelope for protection. It can provide the full functionality of SP4, although not fully utilized by the SP4E Entity. The SP4 Machine submodule uses the interface to the Key Manager module to request, replace, return, and verify traffic key assignment. Assigned keys are used, in turn, through the interface to the Cryptographic Facility to encipher/decipher information and compute integrity check values. The Timer Facility submodule provides the SP4 Machine with a mechanism for determining protracted delays with the Key Manager. The two submodules communicate with each other through an internal interaction point.

When either an incoming request from the user or an incoming indication from the network occurs respectively at a PNSAP or NSAP, the SP4E Entity instantiates both a SP4 Machine submodule and a supporting timer submodule, connecting them together through the "T" interaction point. The parent SP4E Entity must multiplex subsequent events that occur at either the PNSAP or matching NSAP, to and from the corresponding "P" and "N" interaction points of the child SP4 Machine, to allow the child to assume responsibility for these interactions. Similarly, the SP4E Entity also multiplexes events for the child SP4 Machine's "K" interaction point over the "key service" interaction point with the Key Manager. After dealing with the triggering event, the submodules remain in place to service future events that occur at the PNSAP and NSAP pair. Figure 2 illustrates an instance of two, peer SP4E Entities and their submodules.

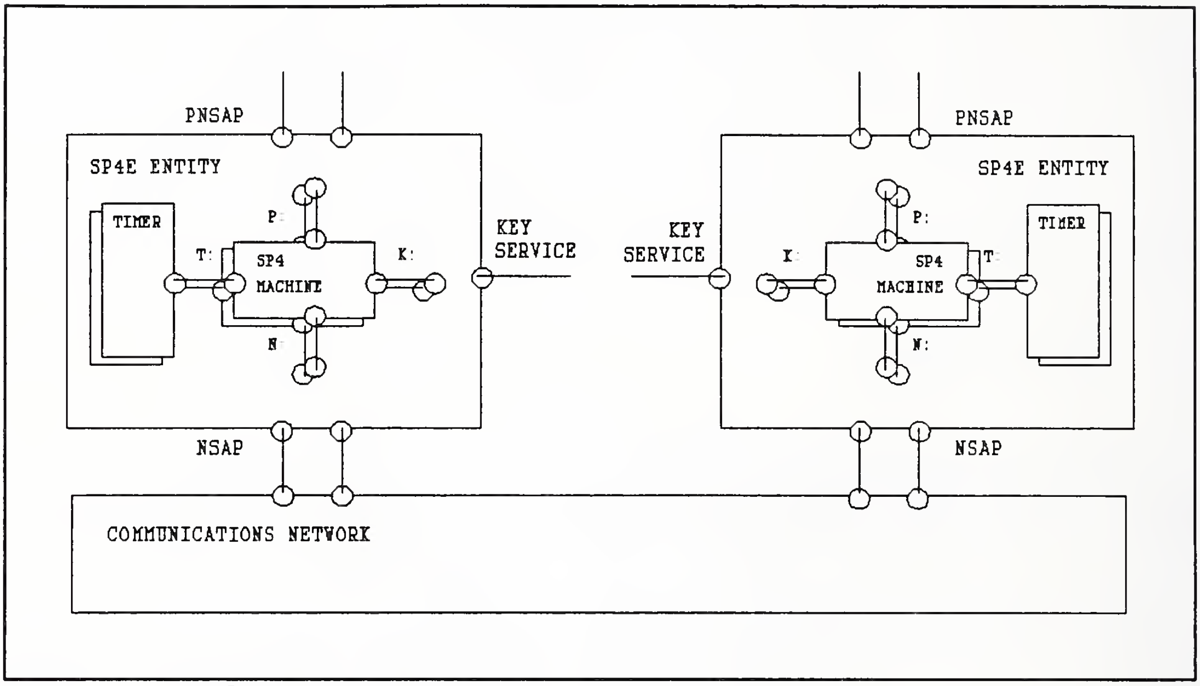

Figure 2: Peer SP4E Entity Modules

2.3 The Key Manager

The Key Manager module retains responsibility for establishing security associations for the SP4E Entity module, as mandated by the security policy parameters and access control restrictions contained within a Security Management Information Base (SMIB). It provides a key service interface corresponding to that used by the SP4E Entity module. The Key Manager uses its interface to the Cryptographic Facility to establish and manage traffic encryption keys. It exchanges key management information directly with its peer through a pair of initiator/responder interaction points.

The Key Manager module is composed of a pair of submodules that conduct the activities of either an initiator or responder key management agent. The Key Manager instantiates a KM_Initiator submodule for each service request received from an SP4E Entity. It instantiates a KM_Responder submodule when a key management primitive is received from an KM_Initiator submodule of a peer Key Manager. Once a submodule is instantiated it assumes responsibility for the triggering event and any subsequent event interactions. Subsequent interactions between communicating peer KM_Initiator and KM_Responder submodules are multiplexed, from either the "I2R" (initiator-to-responder) or "R2I" (responder-to-initiator) interaction points over the corresponding "to peer" and "from peer" interaction points, by the respective parent Key Manager modules. Similarly, the parent Key Manager also multiplexes events for the child's "K" interaction point over the "key service" interaction point with the SP4E Entity. Figure 3 illustrates an instance of two, peer Key Manager entities and their submodules.

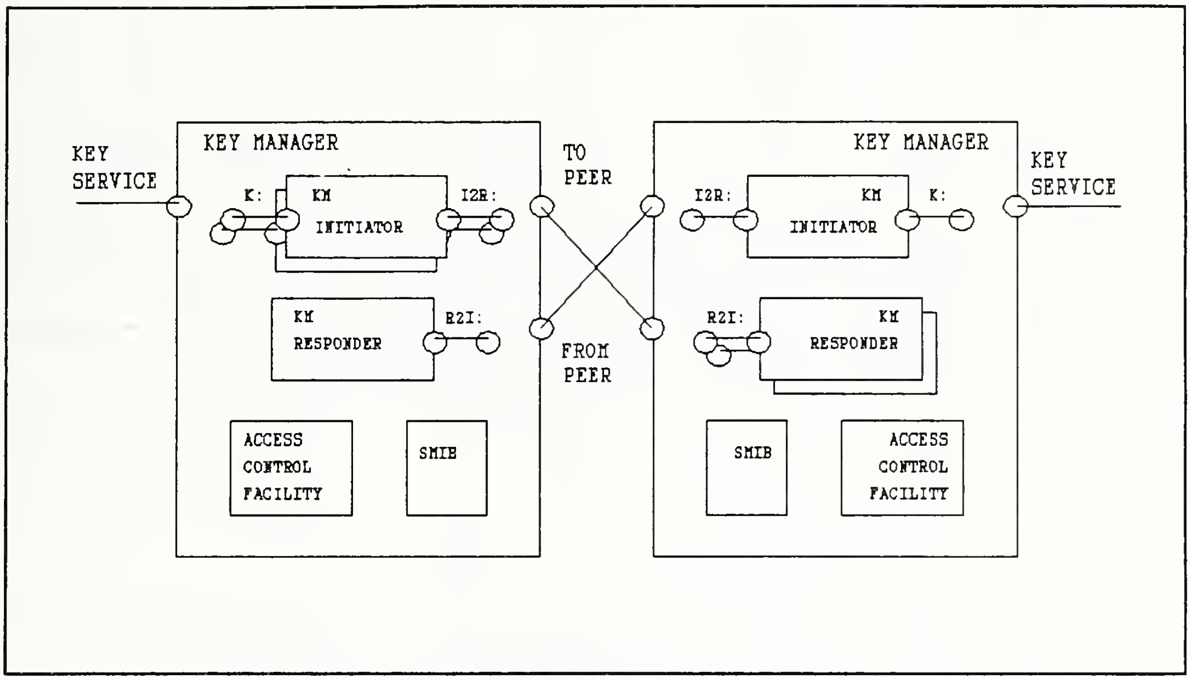

Figure 3: Peer Key Manager Entity Modules

A KM_Initiator submodule must originate the dialogue to establish and manage a security association, while a KM_Responder submodule residing in a peer Key Manager merely reacts to the KM_Initiator's requests. Since communicating KM_Initiator and KM_Responder submodules reside within different Key Managers, each may be representing end-systems with different security policies. Each would employ a potentially different SMIB, and must therefore be capable of negotiating a common security position. To aid in the negotiation, an Access Control Facility is defined. The Access Control Facility consists of a group of functions and procedures that collectively determine access control permissions.

Like the Access Control Facility, a group of function and procedure definitions makes up the interface to the SMIB. Collectively they provide the capability to create new objects, to manipulate existing objects that populate the SMIB, to enter, access, or modify information stored as attributes of those objects, and to destroy objects no longer needed. The interface operates at the object level. It is used exclusively by the Key Manager and its submodules, who are relied upon to maintain the integrity of the SMIB.

2.4 The Communications Network

The Communications Network module represents the lower three layers of the ISO reference model for OSI. It provides a connectionless Network service that supports the exchange of data between peer SP4E entities. The Communications Network module also displays network traffic as part of the demonstration capability, using the Man-Machine Interface Facility.

2.5 The Cryptographic Facility

The Cryptographic Facility houses the cryptographic algorithms needed by the security protocol entities to provide security services. It is intended to isolate the cryptographic algorithms from the protocol machines that employ them. To function properly for the security protocol entities, the Cryptographic Facility must contain appropriate keying material, established by the Key Manager. Therefore, the facility supports a bipartite interface: one portion for the SP4E Entity and the other for the Key Manager. Figure 4 gives an illustration of the Cryptographic Facility.

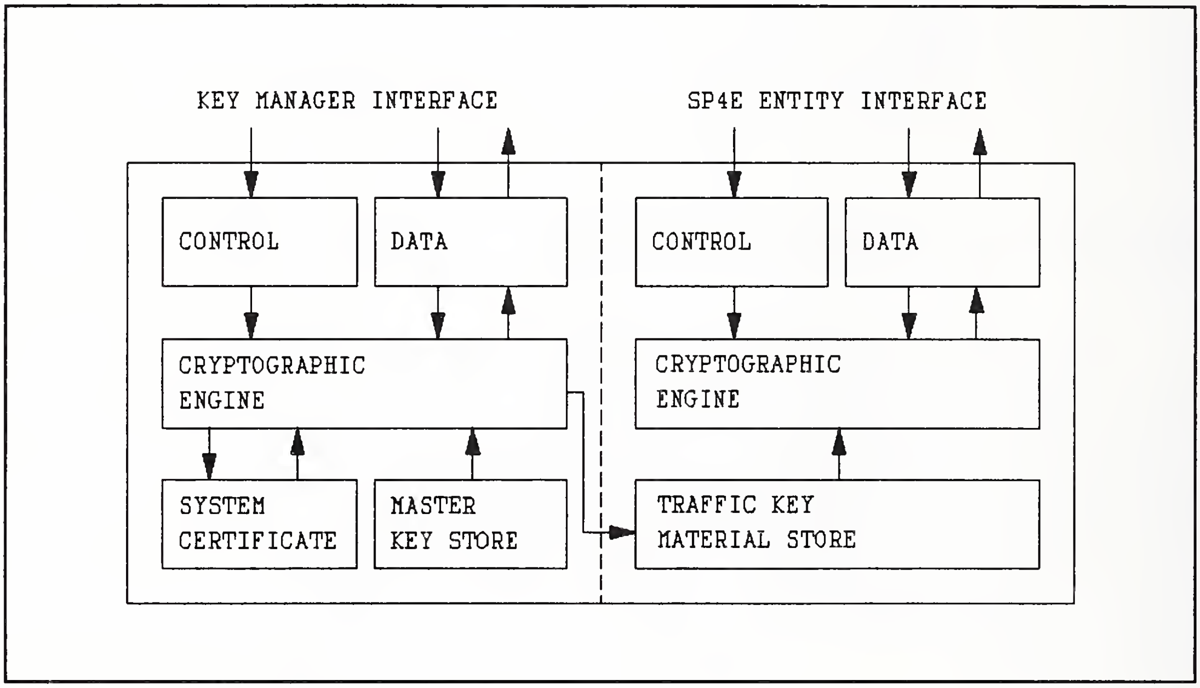

Figure 4: The Cryptographic Facility

The SP4E Entity portion of the interface provides for the generation of message authentication and manipulation detection codes, encryption/decryption of protocol data units, and status checking of keying material. This portion of the interface is relatively independent of the class of cryptographic system in which the underlying algorithms fall. The Key Manager portion of the interface, however, is dependent on both the cryptosystem and key management scheme employed. For this specification, the interface supports a symmetric cryptosystem with an asymmetric, credential based key management scheme. The Key Manager portion of the interface provides for the initialization of the Cryptographic Facility, the loading of system certificates, the generation of traffic keys and system credentials, the loading of traffic keys, and the release of established traffic keys.

3. DESIGN COMMENTS AND HIGHLIGHTS

The sections that follow highlight implications of the design choices made and indicate potentially troublesome areas in the interpretation of the SP4 standard. The discussion is wide ranging and includes topics concerning service and protocol functionality, the formal description technique, and the structure of the design.

3.1 SP4E Profile

Table 1 summarizes the capabilities of the SP4E subset in comparison with the full range of security mechanisms employed by SP4. The table shows the name of the security mechanism, the paragraph reference to the SP4 standard [11], and whether the mechanism is mandatory (M), optional (O), or not applicable (NA) from the point of view of both an initiator (I) and responder (R). The table is divided into two categories of mechanisms, basic and optional, that respectively indicate if the security mechanism is inherent to SP4 or can be negotiated.

TABLE 1: SP4E Capability Profile

MECHANISM	REF	I	R	CMTS
Basic Mechanisms:				
Padding	6.6	M	M	
Concatenation/Separation	6.1	M	M	
Encapsulation/Decapsulation	5.5	M	M	
Address Verification	6.4	M	M	
Connection Integrity	5.5.2			
direction ind	6.3.2	M	M	
transport s/n	5.5.2	M	M	1
Optional Mechanisms:				
Encipherment	6.2	O	M	2
Data Integrity Check	6.3.1	O	M	2
Key Granularity	5.1	O	M	3
Connection Integrity	5.5.2			
per tc granularity	5.1	NA	NA	3,4
unique s/n	6.3.3.1	NA	NA	1,5
final s/n	6.3.3.1	NA	NA	1,5
connection release	6.7	NA	NA	5
Security Labels	6.5	O	M	

(1) Only mandatory for class 2, 3, or 4 Transport Protocol; otherwise, not applicable.

(2) At least one of these mechanisms must be negotiated

for an association.

(3) SP4E cannot provide protection on a "per Transport connection" basis.

(4) Not a negotiable SP4E service.

(5) Service provided only when individual, "per Transport connection" key granularity negotiated.

From the table entries, the limitations of SP4E versus SP4C functionality are clearly visible. The SP4E limitations lie directly with the connection integrity mechanisms not applicable to SP4E due to the key granularity choices available. That is, several of the listed security mechanisms are applicable only to individually protected Transport connections and are therefore not supportable by SP4E.

3.2 Design Limitations

3.2.1 SP4E External to Transport

The choice of modeling SP4E external to and independent from Transport introduces some limitations on the protection operations performed. The problem lies with the inability of the Transport entity operating through the standard Network service interface offered by the SP4E Entity, to convey the proper security context for its operations. The interface parameters it lacks include the type of Transport Protocol data unit (TPDU) submitted, the desired security protection, the associated security label, and for connection oriented Transport, the connection identifier. Even if the service interface is extended to include such parameters, it is doubtful that a typical Transport entity would be capable of supplying them, since the standard Transport service interface itself does not include such detailed, upper level, quality of service (QOS) information.

The SP4E specification sidesteps this issue somewhat, by obtaining the security context information directly through its interface to the Key Manager, as part of the negotiated attributes for a security association. This approach has the flexibility to accommodate protection QOS parameters when available at the standard Network service interface, since this information is passed along to the Key Manager in a request. Should protection QOS parameters not be available, the Key Manager may be able to determine the appropriate security context from other sources, such as the SMIB or the security service interface of the operating system. In either circumstance, the same security mechanisms are applied uniformly to all traffic between a source/destination address pair at a specific security level.

3.2.2 Connectionless Network Interface

The use of the connectionless Network service interface for the SP4E Entity poses an additional challenge for the specification. The connectionless service constrains how the duration of an instance of communications at a specific security level, for a source/destination address pair, is determined, since the opening and closing are not explicit at the service interface. The SP4E Entity must indirectly determine utilization by noting when activity begins and when any significant lulls in activity occur. A connection oriented interface would simplify this task, since the opening and closing of a connection explicitly indicate the beginning and end of the utilization period.

For the specification, the SP4E Entity is required to maintain a list of source/destination address pairs and security level designator for management of SP4 Machine instances and their associated traffic key utilization. The SP4 Machines are responsible for interacting with the Key Manager to obtain and relinquish traffic keys. The final decision whether to release a security association or maintain it for possible reuse on a subsequent key request is left to the Key Manager.

3.2.3 Directly Connected Key Managers

The SDNS architecture requires key management functionality to be collocated with other applications within an end-system, whenever a lower layer security protocol is employed [ref. 13]. Although logically collocated, the communication services afforded it are distinct to allow implementation in physically separate hardware for high security environments. In such arrangements, key management may be considered as a hidden internal host. For SP4, the network selector (NSEL) portion of the Network address is intended to be used to locate the appropriate communications protocol stack. Figure 5 illustrates a dual stack model used to represent the partitioning of key management communication services from normal user communication services.

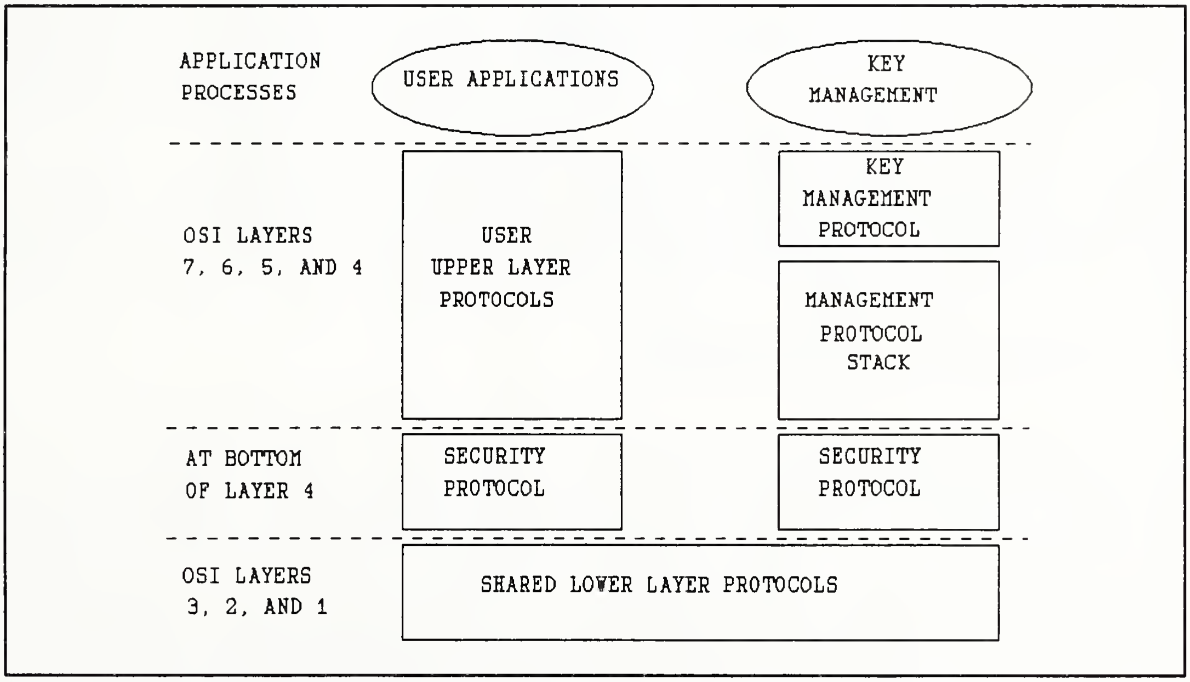

Figure 5: Dual Stack Model

The specification represents the two distinct communications services in different ways. SP4E Entity modules communicate with one another through the interface offered by the Communications Network. Key Manager modules communicate directly through a pair of peer interaction points, rather than use the Communications Network module as might be expected. Since the focus of the specification is the security protocol, key management functionality is treated as part of the test harness, and therefore not specified to its full extent. As an Application Layer Protocol, complete specification of key management would require modeling a full upper layer protocol stack, as well as defining the interface to the support tools of an abstract syntax notation (ASN.1) parser for encoding/decoding operations. In their current form, the Key Manager modules depend on only the Estelle specification, and are easily modified to accommodate changes or enhancement.

3.2.4 Estelle Idiosyncrasies

There are a couple of peculiarities in Estelle that should be noted since they affect the form of the specification. As mentioned earlier, Estelle uses interaction points to model the interactions between modules. Interactions operate asynchronously and are well suited to the modeling of communications protocols. Unfortunately, there is no counterpart in the language for modeling synchronous operations between modules. Although functions and procedures allow for modelling of synchronous operations, their scope is limited to that of the defining module. Furthermore, there is no mechanism for packaging a collection of related functions, procedures, and data types to be collected into a module for export to other modules that require them.

This limitation affected the modeling of the Cryptographic Facility and the Access Control Facility. For the Cryptographic Facility, all associated functions and procedures are defined as primitive routines, and located at the lowest level in the specification where they are referenced by the Key Manager and SP4E Entity modules that use them. Similarly, the functions and procedures of the Access Control Facility are defined as primitive and split between the Key Manager submodules that employ them. Although a module definition would give a better conceptual picture of the facility, the method used is typical of facilities described in Estelle.

3.3 Security Troika Responsibilities

It is important to recognize that the SP4E Entity is one member of a troika responsible for security, illustrated in Figure 6. The other members shown are the Key Manager and the Cryptographic Facility. Delegation of responsibilities between members of the troika is pivotal to the expression of the specification. For example, a lessening in services offered by the Cryptographic Facility to the SP4E Entity could require enhancement of the service interface to SP4E, offered by the Key Manager. Since the SP4 standard does not define its external interfaces in precise detail, expressions of SP4 are expected to differ in this area.

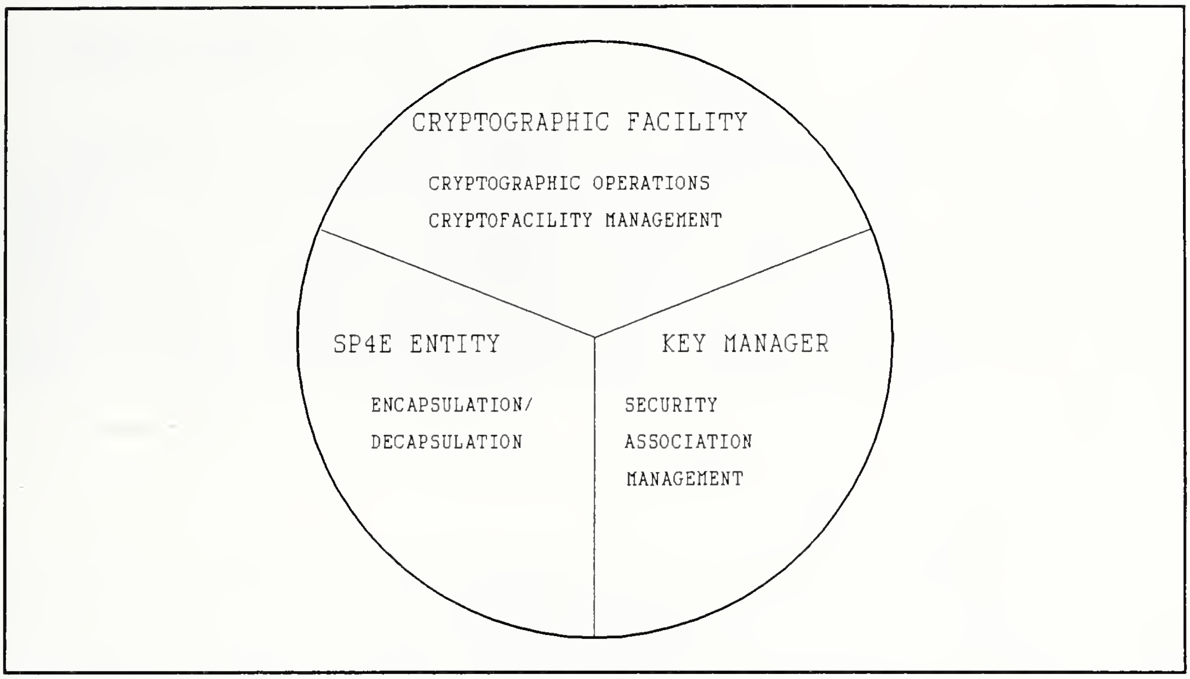

Figure 6: Security Troika

3.4 Key Management Scheme

A credential based key management scheme is used, based on a subset of the SDNS key management protocols. The primitives supported allow for exchange of credentials, negotiation of security attributes for a traffic key, and replacement of an expired traffic key by another. The specification assumes manual distribution of the device authentication certificates used to form credentials, that are subsequently used within traffic key establishment protocol exchanges.

The key management scheme is bound closely with its interface to the Cryptographic Facility, and at least one object in the SMIB, the system certificate. Another choice of key management scheme would require changes to these areas, and perhaps the Access Control Facility interface. However, the interface to the SP4E Entity would most likely not be affected by such a change.

3.5 Access Control Facility

Access Control Facility is modeled as a component internal to the Key Manager. It localizes access to the SMIB for information related to the security associations permitted and to any further restrictions that may apply. The Access Control Facility stipulates the envelope of operations for a newly created key. It is the responsibility of the Key Manager to enforce those boundaries during the negotiation of the traffic key security attributes. The operation of the Access Control Facility is closely associated with the key management scheme utilized as well as the data objects modeled within the SMIB.

The interface to the Access Control Facility is modeled as a set of routines that convey security attribute information associated with a proposed security association. The SP4E Entity does not directly use the Access Control Facility, but receives its information indirectly through the Key Manager. Access control decisions for the specification are based primarily on the information contained in the SMIB. As noted earlier, the specification allows an SP4E Entity to pass security options accepted from the Transport entity onto the Key Manager, and to have those options considered by the Access Control Facility. However, since the Network quality of service parameter for protection is currently limited in scope to the use of integrity and confidentiality security services, the consideration that the facility can give to the Transport entity's request is restricted to those areas.

3.6 Cryptographic Facility

The Cryptographic Facility provides an interface based on the use of an algorithm identifier to select a choice of cryptographic algorithm from the set supported by the facility. This feature allows the cryptographic algorithms to be independent of the security protocols that employ them, consistent with the SDNS architectural principles. Similarly, a key identifier, rather than a location identifier, is used to reference key material within the Cryptographic Facility.

The specification gives the responsibility of maintaining the cryptographic period for all loaded key material to the Cryptographic Facility. This allows inappropriate operations to be blocked whenever the cryptographic period expires. For example, decryptions may be performed, but encryptions are not allowed. The status of the cryptographic period for key material with a given key identifier can be obtained through the interface.

3.7 Alarms and Event Reporting

The top level of the design does not include a mechanism to log security events or raise security alarms. Modeling of these areas is postponed to a lower level of detail, in the specification of the individual protocol machines. These areas are considered part of systems management, and intended to comply with the draft ISO network management document on the security alarm reporting function [ref. 10]. This function supports the indication of several security alarm types and associated severity levels.

3.8 Trust

The SP4E specification by its very nature attempts to be free from any specific hardware/software implementation restrictions. Although concerned with correctness of operation, the specification itself does not provide any guarantee that a system implemented accordingly would be secure. That assurance can be gained only by analyzing the actual implementation and evaluating it according to an established criterion.

3.9 Administration

Throughout the specification and in particular with the SMIB, the aspect of system administration is ignored. It is assumed that at the beginning of execution the SMIB contains the correctly represented values of intended configuration and policy attributes of objects within it. The maintenance of SMIB objects and attribute values is outside the scope of the specification, as is the modeling of network management entities. In practice, the SMIB may be implemented using a database management system and administered through the data definition and data manipulation facilities provided.

4. SP4E ENTITY SUBMODULES

The submodules of the SP4E Entity identified in the top level design are the SP4 Machine and the Timer Facility. They are illustrated in Figure 7 below. The sections that follow review each submodule in detail. The corresponding Estelle specification appears in Appendix B.

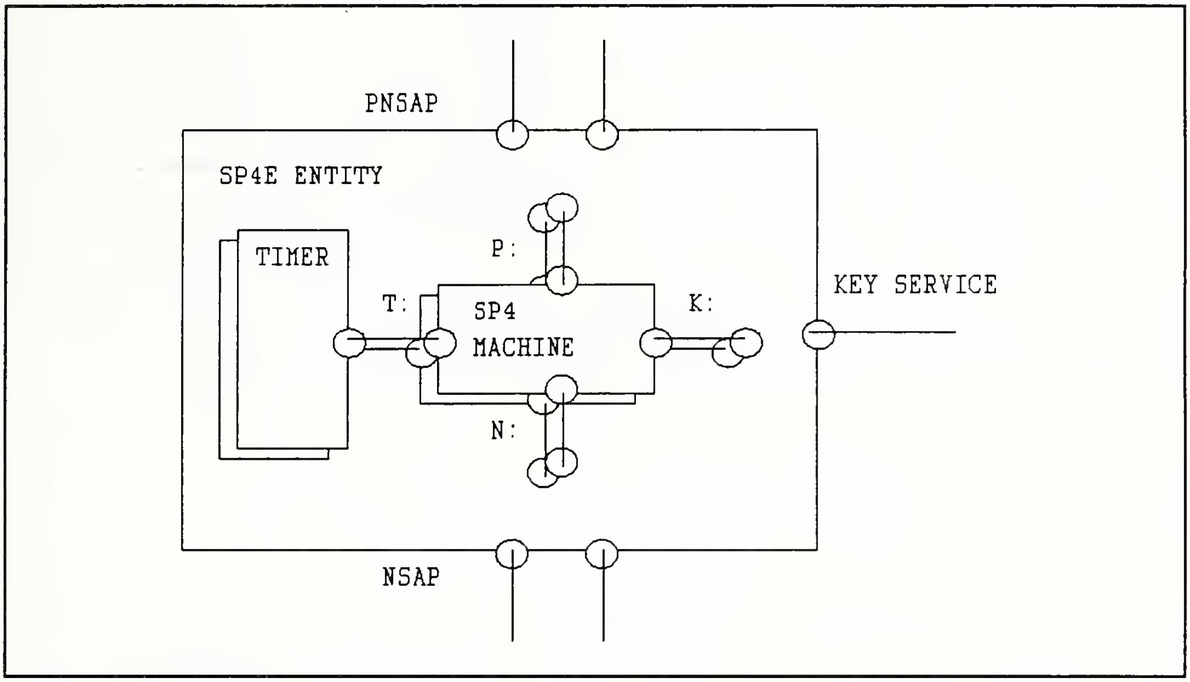

Figure 7: SP4 Entity Submodules

4.1 The Timer Facility

The Timer Facility is a simple mechanism to keep track of elapsed time. It supports two distinct timers: the key_manager and the inactivity timers. A timer may be set or cancelled, and when it expires, outputs an alarm. Each operation pertains to an individual timer, except cancellation, which effects both timers.

The SP4 Machine uses the key_manager timer to track the progress of the Key Manager and take action whenever an unreasonably long period of time elapses without response to a request. Similarly, the SP4 Machine uses the inactivity timer to decide when to request termination and release of a security association. The Timer Facility is easily modelled with the Estelle language constructs that support the semantics for delay representation.

4.2 The SP4 Machine

The SP4 Machine module is the cornerstone of the formal description. It encompasses the entire set of security mechanisms mandated by SP4. It could therefore be directly used as a child module from within another parent module, such as the Transport or Network entity, to bind the security mechanisms it offers more closely to the resources it protects. By integrating a SP4 Machine module internally, finer degrees of control over its mechanisms can be applied by the parent.

From within the SP4E Entity, a SP4 Machine module is instantiated at a given security level for each pair of active source and destination addresses. Activity is indicated by events that occur at either a pseudo Network service access point or a Network service access point, respectively through either a Transport User unit_data request or a Communications Network unit_data indication. The SP4 Machine is instantiated in either an initiator or responder mode, corresponding to whether a Transport or Network event triggered the action. When activity ceases, the SP4 Machine is terminated. The finite state machine for the SP4 Machine module is illustrated in Figure 8.

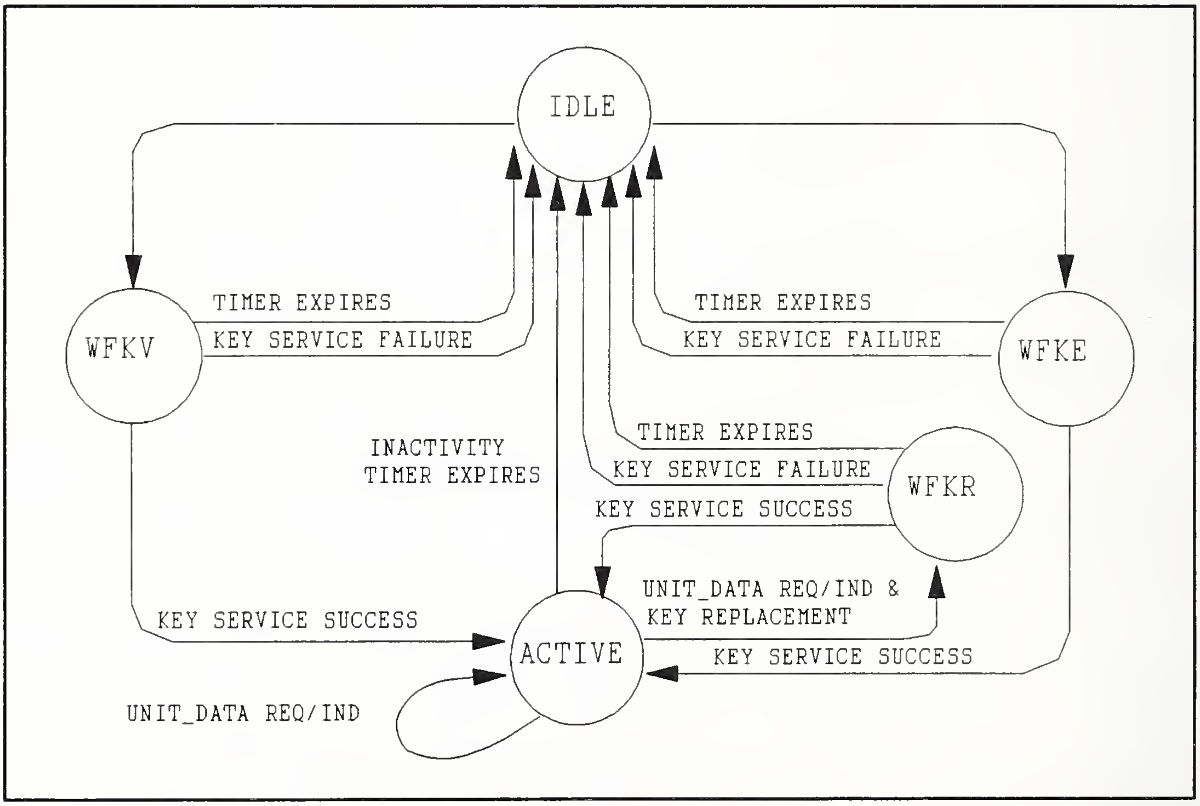

Figure 8: Finite State Machine for SP4_Machine Module

The following items are either assumed or committed in the SP4 Machine submodule specification:

(1) Incoming Network service data units (NSDUs) from outside the SP4 Machine may contain only a single Transport Protocol data unit (TPDU). No concatenated TPDUs are expected.

(2) Outgoing NPDUs from inside the SP4 Machine contain only a single TPDU. Concatenation of encapsulated TPDUs is not performed.

(3) Determination of whether key replacement is required, does not take place in any transition immediately following that in which a key is either established or verified with the Key Manager. It is assumed that the Key Manager would not offer or accept a key that is about to expire.

(4) The SP4 Machine applies layer flow control to incoming SDUs from both the Network and Transport entities, when awaiting a response from the Key Manager.

(5) A SP4 Machine maintains only a single level of labeling based on the quality of service request of the user, the constraints imposed by the Access Control Facility, and the Key Manager negotiation with its peer.

(6) Security associations may be established prior to and persist beyond the lifetime of a SP4 Machine, and in the most general case be simultaneously shared by several SP4 Machines.

(7) After the establishment of a security association, the Key Manager will interpret all subsequent SP4 Machine interactions according to the key granularity initially negotiated with the peer Key Manager.

(8) The SP4 Machine uses a common routine to log and report security events. Five security alarm types and four severity levels may be indicated following the draft ISO network management document on the security alarm reporting function [ref. 10].

(9) The following conditions detected by the SP4 Machine are assigned the security event severity levels and alarm types indicated:
   (a) timeout of the Key Manager - critical time domain violation;
   (b) failures of key service request - major security domain violation; and
   (c) failures encountered wrapping/unwrapping Transport Protocol data units - major security domain violation, time domain violation, or integrity violation.

4.2.1 Initiator Mode

The finite state machine is initialized to the IDLE state. If instantiated in the initiator mode the first transition is to the wait for key establishment (WFKE) state. During the transition, the key_manager timer is set and a request for establishment of a key issued to the Key Manager. If the timer expires before a response is returned from the Key Manager, a key_manager timeout is reported, the input data discarded, and a request for termination issued during the transition back to the IDLE state. Similar actions are taken when in the IDLE state and a negative response is received from the Key Manager. If however, a positive response is received from the Key Manager, the SP4 Machine proceeds to encrypt the Transport Protocol data unit (TPDU), encode the SE TPDU envelope, submit the result to the Communications Network, and set its inactivity timer during the transition to the ACTIVE state.

Once in the ACTIVE state, the SP4 Machine handles all incoming requests from the user and indications from the network, while remaining in this state. Transition out of the ACTIVE state may occur if the cryptographic period elapses and key replacement must occur, or if no activity on the source/destination address pair occurs and the machine spontaneously requests termination of itself. (Note that for an embedded SP4 entity, transition out of the ACTIVE state would occur upon receipt of a disconnect request.) During the transition from the ACTIVE state to the wait for key replacement (WFKR) state, key replacement actions are taken similar to those made during the transition from the IDLE to the WFKE state. That is, the key_manager timer is set and a request for key replacement issued to the Key Manager.

4.2.2 Responder Mode

The actions of the SP4 Machine parallel those described above, when it is instantiated in the responder mode. The transition from the IDLE state to the wait for key verification (WFKV) state occurs spontaneously. During the transition, the key_manager timer is set and a request for verification of the received key identifier is issued to the Key Manager.

Once in the WFKV state three events may occur, two bad and one good. If the key_manager timer expires before a response is returned from it, a key_manager timeout is reported, the input data discarded, and self termination requested during a transition back to the IDLE state. Similar actions are taken during the transition to the IDLE state, if a negative response is received from the Key Manager. If a positive response is received from the Key Manager, the SP4 Machine can proceed to decode the SE TPDU, decrypt the enclosed TPDU, and deliver the result to the Transport User, during the transition to the ACTIVE state. Once in the ACTIVE state processing occurs as described earlier for the initiator mode.

5. KEY MANAGER SUBMODULES

The submodules of the Key Manager identified in the top level design are the KM_Initiator, the KM_Responder, the Access Control Facility and the Security Management Information Base (SMIB). They are illustrated in Figure 9. The sections that follow review each submodule in detail. The corresponding Estelle specification appears in Appendix C. Note however, the formal specification contains only a subset of the transitions described in this chapter. Only those transitions that are not dependent on exchanges between peer manager entities to establish and manage keys, are included. The specification relies totally on pre-established information being available within the SMIB.

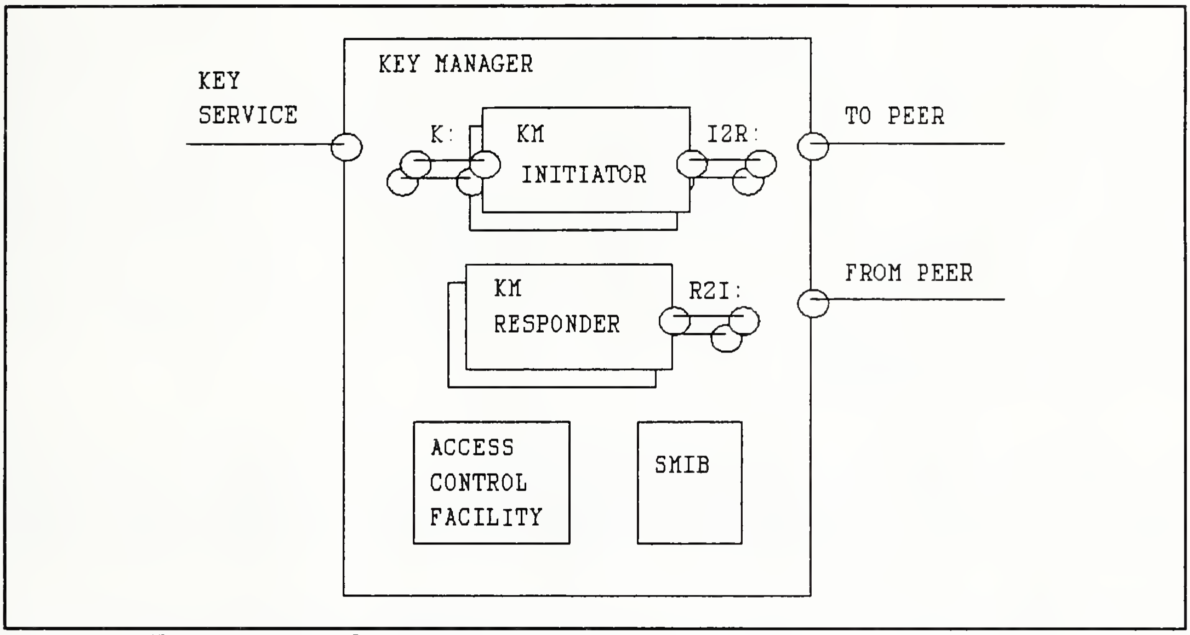

Figure 9: Key Manager Submodules

The following items are either assumed or committed in the specification of the Key Manager submodules:

(1) Error free exchanges are assumed between peer, Key Manager submodule entities, and between corresponding SP4 entity and Key Manager submodules.

(2) All security associations represented within the SMIB are assumed to be at a single security level.

(3) A pair of Key Managers corresponding to two individual end-systems is modelled. Therefore, only a single key management stream is utilized. However, the specification may be configured to permit multiple Key Managers corresponding to multiple end-systems.

5.1 KM_Initiator

The KM_Initiator is instantiated by its parent to handle a request from the corresponding SP4 Machine. There is, at most, one KM_Initiator module active for each SP4 Machine. As mentioned previously, error free exchanges between peer Key Manager entities (and with its corresponding KM_Responder submodule) are assumed. Therefore, no timeouts are set or any unexpected or abnormal conditions accommodated in the specification. For simplicity, a KM_Initiator module is instantiated for each key service request and is terminated upon the issuing of a corresponding response. Figure 10 illustrates the finite state machine for the KM_Initiator module.

The KM_Initiator is initialized to the IDLE state. Upon receipt of a request for key establishment from a SP4 Machine it transitions to either the IDLE or wait for peer authentication (WFPA) state depending respectively on whether an appropriate key exists in the SMIB, or needs to be established. In the former case, a positive key service response is issued to the SP4 Machine, the Cryptographic Facility initialized with the appropriate key material, the SMIB updated accordingly, and a request for termination made during the transition. In the latter case, the transition to the WFKA state is taken, whereby a request to exchange credentials is sent out to a peer KM_Responder entity, as described in the next section.

Once in the WFPA state, if a negative response is received from the corresponding KM_Responder, a negative key service response is issued to the SP4 Machine, termination requested, and a transition taken to the idle state. If, on the other hand, a positive response is received, the KM_Initiator obtains the proposed attributes from the Access Control Facility, sends them out in a request for security attributes, and transitions to the wait for access control (WFAC) state to await the response.

Upon receipt of the response to the security attributes request, a transition is made from the WFAC to the IDLE state. During the transition a positive key service response is issued to the SP4 Machine, the Cryptographic Facility initialized with the appropriate key material, the SMIB updated accordingly, and a request for termination made.

The handling of the SP4 Machine request for key replacement occurs from the IDLE state and parallels that of the request for key establishment. The main differences are in the checking and actions that occur during the transition from the idle state. That is, a local key must already exist for key replacement, the SMIB and Cryptofacility updated if appropriate, and a request for key deletion sent out to the peer Key Manager entity if appropriate.

The key service request for verification is handled during a single transition from the IDLE state to itself. During the transition the SMIB is checked, response information retrieved and issued to the requesting SP4 Machine, and termination requested. The key

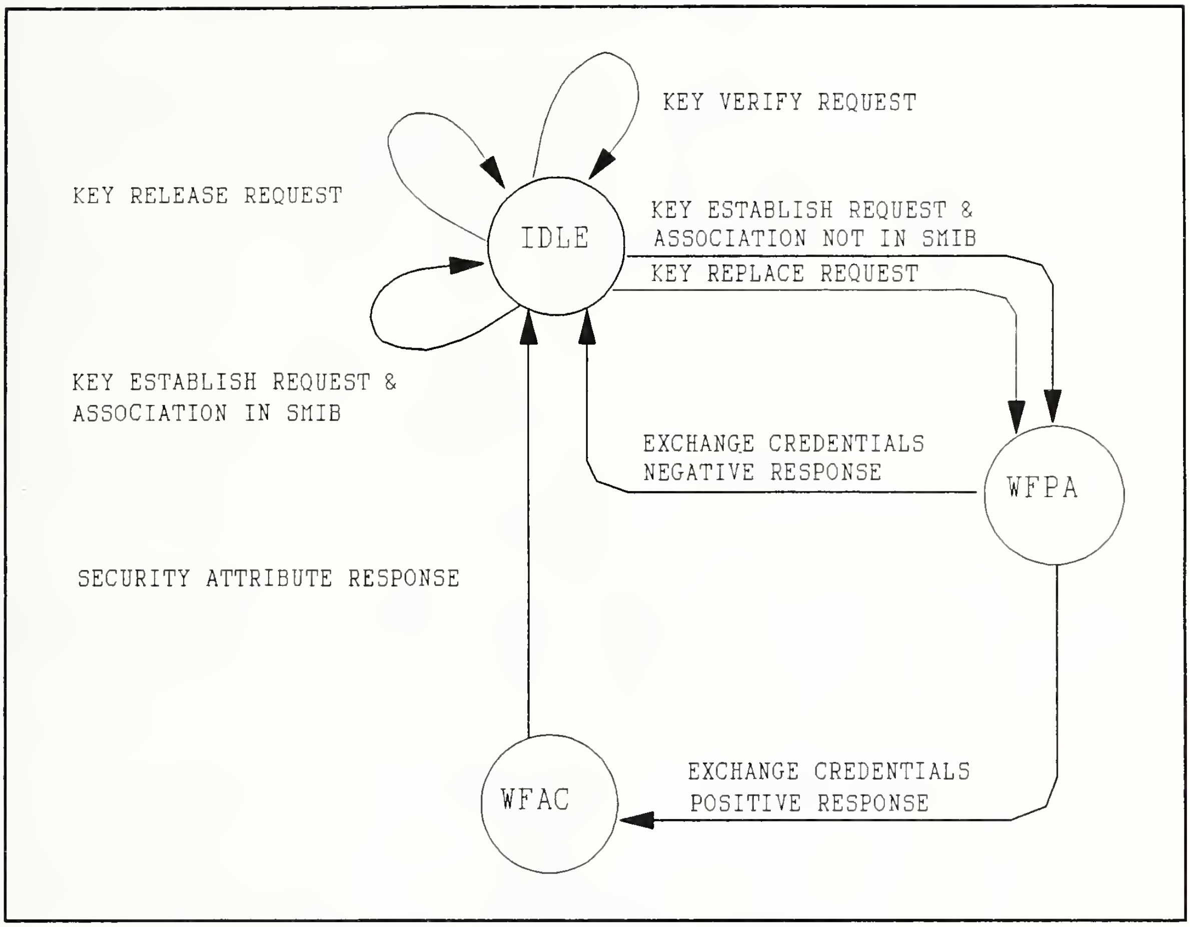

Figure 10: Finite State Machine for KM_Initiator Submodule

service request for release is also a single transition from the IDLE state to itself that merely updates the SMIB and possibly the Cryptographic Facility.

5.2 KM_Responder

The KM_Responder is the peer entity of the KM_Initiator. It is instantiated by its parent whenever an appropriate key management request is received from the corresponding KM_Initiator, and remains until the entire security association negotiation is complete. The two state, finite state machine for the KM_Responder is illustrated in Figure 11 that follows.

The KM_Responder is initialized to the IDLE state to handle a request to exchange credentials. There are three transitions that can occur from the IDLE state. The first and simplest transition is when the credentials received from the peer are invalid. Here a negative response can be sent out and termination requested during the transition back to the IDLE state. The second case is also simple and occurs when a request for key deletion is received from the peer. During the transition back to the IDLE state both the

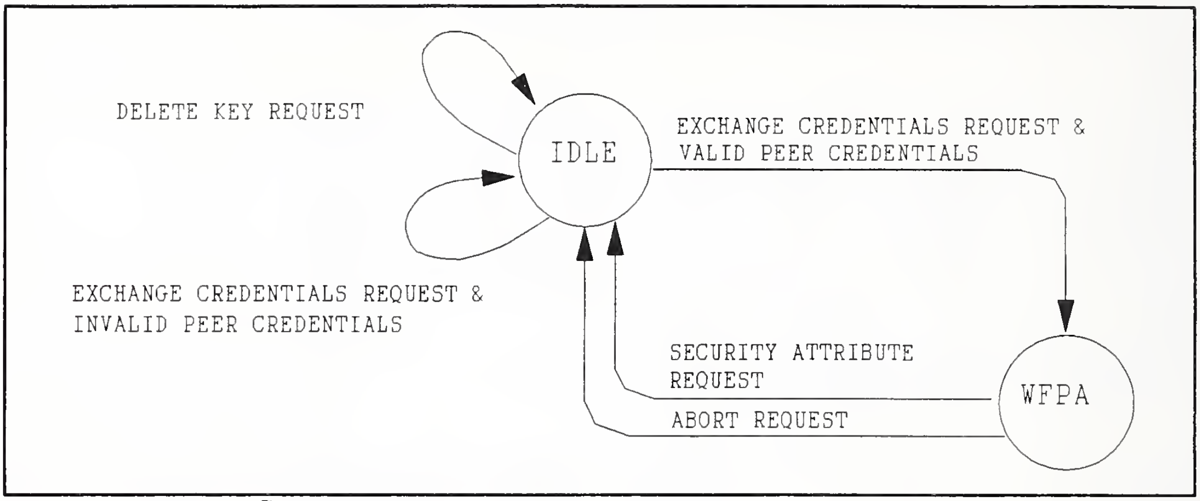

Figure 11: Finite State Machine for KM_Responder

SMIB and Cryptofacility are updated accordingly and termination requested. The final case occurs if the peer credentials are valid. In this situation a response to exchange credentials is issued and a transition is taken to the wait for peer authentication (WFPA) state to await a follow on request.

The follow on request expected is either a request for security attributes, or an abort request. The former occurs if the local credentials were valid at the peer, and the latter if they were invalid. A response for security attributes is generated using the Access Control Facility and issued in reply to a request for security attributes, during the transition to the IDLE state. The response may be positive or negative depending on the results obtained from the Access Control Facility. An abort request is handled simply by requesting termination during the transition back to the idle state. Note the abort request is used to model the A-Abort access control service primitive that would normally be used by a layered entity.

5.3 Access Control Facility

The Access Control Facility is represented as a collection of Estelle functions and procedures, intended to localize the authorization and access control determination process. The routines follow the guidelines given in the SDNS access control concept document [ref. 11]. The Access Control Facility routines make direct use of the SMIB interface to carry out their actions. Although collectively they form a facility, they are not represented as a module nor bound together in any way, due to limitations in Estelle to support this type of modelling. The routines are independent of any interaction point events and therefore not described in terms of a finite state machine.

5.4 Security Management Information Base (SMIB)

Like the Access Control Facility the SMIB is also a collection of related functions and procedures, whose actions are modelled independent of any interaction point events. The SMIB localizes the handling of requests for management information related to security. The contents of the SMIB are retained between incarnations of the system specification. The Key Manager must maintain the integrity of the information throughout its operation.

6. SMIB DATA MODEL

The SMIB contains the set of security management information needed for SP4E operation. Security management information is specified in terms of object classes. An object class represents a view of the required security information for modelling purposes. An object is defined to be an instance of an object class. The characteristics of an object class are defined by its attributes that form the template for object instances. For simplicity, the term "object" is used synonymously for "object class" throughout this chapter.

The SMIB is conceptually organized into three portions: authentication, authorization, and access control. The authentication portion contains information that may be used to establish the identity of the system. The authorization portion includes information concerning the granting of access permissions between authenticated end-systems. The access control portion of the SMIB contains information concerning the enforcement of access control between end-systems.

Figure 12 gives an overview of the data model for the SMIB that illustrates the relationships between security objects. It is expected that these objects would be related to other objects within the more general management information base. Within the figure, a single headed arrow is used to indicate a one-to-many relationship between objects, and a line indicates a one-to-one relationship. The SMIB security objects are elaborated further in the sections that follow. Attributes of all objects are given in Appendix D.

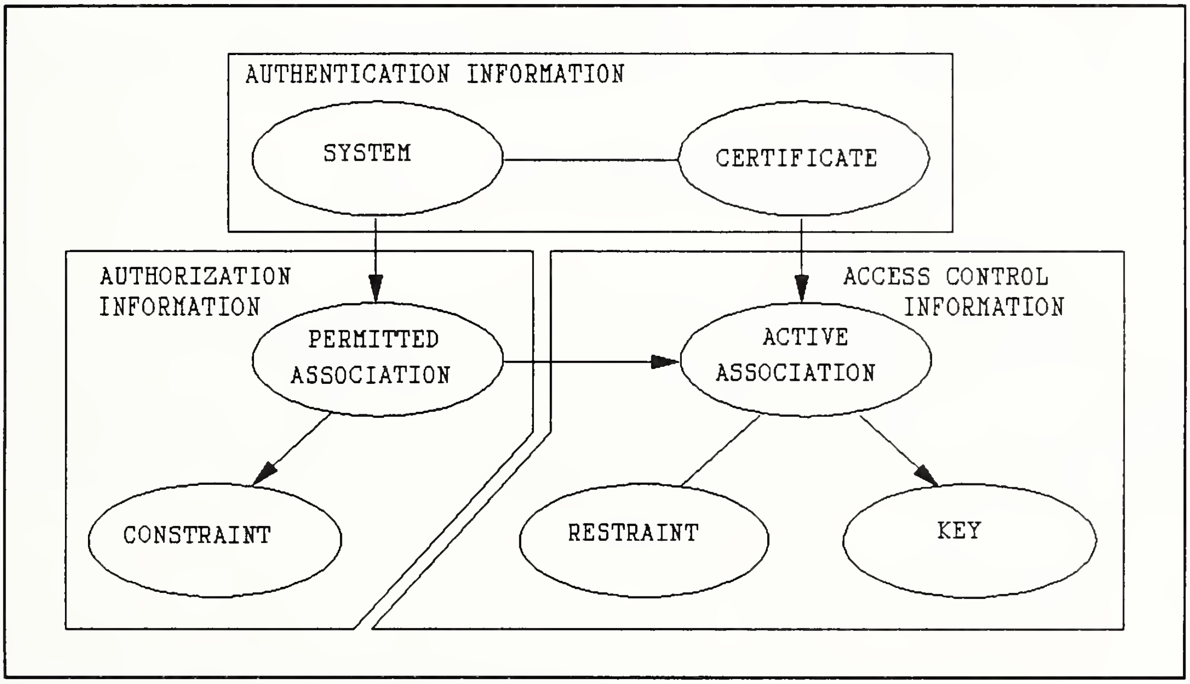

Figure 12: Top Level Data Model

6.1 Authentication Objects

The objects maintained in the Security Management Information Base for authentication purposes are:

(1) the system object, and

(2) the certificate object.

The system object identifies the system and security policy in effect. A common security policy must be established before a collection of open systems may intercommunicate. The policy is a set of rules that mandate whether end-systems may communicate with one another, the criteria for peer authentication, the security mechanisms to be applied to information exchanges, et cetera. The policy itself is reflected in the values of the attributes of objects within other portions of the SMIB, that control the nature and quality of the security associations that can be established.

The certificate object is defined as the basis for peer entity authentication. It is assumed that certificates are distributed by a mutually trusted party for those end-systems wishing to intercommunicate. The data model allows for only a single certificate to be associated with the defined security policy.

6.2 Authorization Objects

The objects maintained in the Security Management Information Base for authorization purposes are:

(1) the permitted association object, and

(2) constraint object.

The permitted association object and constraint object define the envelope of permitted interoperation by indicating respectively which systems may intercommunicate and what security mechanisms must be in effect to comply with the security policy. The constraint object in particular, provides all the information needed to negotiate access control permissions. The attributes of authorization objects pertain exclusively to information required for SP4E interoperation.

6.3 Access Control Objects

The access control objects maintained in the Security Management Information Base are:

(1) the active association object,

(2) the restraint object, and

(3) the key object.

The active association object is central to the dynamic portion of the SMIB. It represents all active security associations that are established and retains the security protection parameters negotiated for the association. It is closely related to the permitted association object and should be maintained in a logically consistent fashion with that representation.

The restraint object provides the information needed for enforcement of the access control restrictions negotiated for an association. It retains the security protection parameters corresponding to an active association object.

The key object contains the keying material used in the cryptographic operations for an active association, and related information regarding the appropriate application of keying material. Note that a single active association may use multiple keys (e.g., during key rollover).

REFERENCES

1. ISO IS 7498, Open systems Interconnection - Basic Reference Model, 1984.

2. ISO IS 7498/2, Open Systems Interconnection Reference Model - Security Architecture, 1988.

3. Specification SND.401, Secure Data Network Systems (SDNS) Security Protocol 4 (SP4), revision 1.3, National Security Agency, May 1989.

4. D. Branstad and others, SP4: A Transport Encapsulation Security Protocol, Proceedings National Computer Security Conference, Sept. 1987.

5. R. Nelson, SDNS Services and Architecture, Proceedings National Computer Security Conference, Sept. 1987.

6. ISO/IEC JTC 1/SC6 N6285, Committee Draft OSI Transport Layer Security Protocol, November 15, 1990

7. ISO/TC 97/SC 21 9074, Estelle: A Formal Description Technique Based on an Extended State Transition Model, June 1988.

8. J.P. Favreau and others, User Guide for the NBS Prototype Compiler for Estelle, Technical Report ICST/SNA - 87/3, NIST, February 1989.

9. W. Majurski, PEDS Reference Manual, Draft Technical Report, NIST, February 1990.

10. ISO/IEC JTC1/SC21 N-4064, Systems Management - Part 7: Security Alarm Reporting Function, 20 November 1989.

11. Specification SND.801, Access Control Concept Document, revision 1.3, National Security Agency, 26 July 1989.

12. U.S. Government Open Systems Interconnection Profile (GOSIP), version 2.0, Federal Information Processing Standard (FIPS) 146-1, April 1991.

13. Paul Lambert, Architectural Model of the SDNS Key Management Protocol, Proceedings of the 11th National Computer Security Conference, October 1988.

APPENDIX A
ESTELLE SPECIFICATION OF SP4E MODULES

This appendix contains the Estelle specification of the SP4E Entity and supporting modules. At this top level of the design, emphasis is placed on the definition of module headers that describe the interfaces between modules, rather than on the definition of the module bodies. The diagram below gives a general overview of the specification that follows.

```
+----Specification---------------------------------------------------------
|
|     +----Transport User module header--------------------
|     |           PseudoNSAP:
|     +--------------------------------------------
|
|     +----SP4E Entity module header-------------------------
|     |           PseudoNSAP:  NSAP:  KEYservice:
|     +--------------------------------------------
|
|     +----Communications Network module header-------
|     |           NSAP:
|     +--------------------------------------------
|
|     +----Key Manager module header-----------------------
|     |           KEYservice:  TOpeer:  FROMpeer:
|     +--------------------------------------------
|
|     +----Transport User module body-----------------------
|
|     +==SP4E Entity module body===========
|
|     +----Communications Network module body---------
|
|     +==Key Manager module body==========
|
+-----------------------------------------------------
```

Interaction point definitions are listed in the diagram for each module header and/or body definition where they appear. The convention of one body definition per module header definition applies throughout the specification. A double line in the diagram indicates that further specification of the module body occurs externally, in either Appendix B or C.

Throughout the specification comments, indicated by enclosing parenthesis-asterisk pairs (i.e., "(* comment *)") or pairs of braces (i.e., { comment }), are used to assist the reader. Several mechanisms are employed to express the specification at a level of abstraction somewhat free from implementation concerns. First, the "..." notation is used to indicate an open data type declaration. Second, the term "primitive" is applied to those functions and procedures, whose details are not expressed as part of the specification.

```
specification TLD systemprocess;

default individual queue;
timescale milliseconds;

(* Global Declarations *)

type
    data_type = ...;  { uninterpreted string of octets }

const
    MAX_NSAP = any integer;
    MAX_K_STREAM = MAX_NSAP;  { one key stream allocated per NSAP }
    MAX_N_ADDRESS  = any integer;  { must be >= 2*MAX_NSAP for simulation }

type
    octet = 0..255;        { one byte }
    name_type = (sp4eentity, sp4machine, keymanager, kminitiator,
                 kmresponder);
    role_type = (stimulate, respond, report);
    phase_type = (start, shutdown, finish);
    smib_id_type = ...;
    cf_id_type = ...;
    vpid_type = ...;

    (* connectionless network related *)
    NSAP_type = 1..MAX_NSAP;
    N_address_type = ...;
    N_QOS_value_type = ...;
    N_QOS_parameter_type = (transit_delay, protection_security,
                            cost_determinants, residual_error_probability,
                            relative_priority, route_selection);
    N_QOS_parameter_set_type = array[N_QOS_parameter_type] of N_QOS_value_type;

    (* key manager related *)
    K_STREAM_type = 1..MAX_K_STREAM;
    kmgr_response_type = (assoc_ok, assoc_failed);
    kmgt_status_type = (trans_ok, trans_failed);
    date_type = record
            month: 1..12;
            day: 1..31;
            year: 0..99;
                 end;
    request_identifier_type = ...;
```

```
    key_identifier_type = ...;
    security_label_type = ...;
    key_attributes_type = ...;
    security_attributes_type = ...;
    algorithm_id_type = ...;

    (* cryptographic facility related *)
    certificate_type = ...;
    credentials_type = ...;
    crypto_status_type = (LOAD_OK, FREE, KEY_IN_PROGRESS, COMPLETE_KEY,
                          BAD_KEY_ID, NO_SPACE);
    cipher_status_type = (MAC_COMPUTED, MDC_COMPUTED,
                          DATA_ENCRYPTED, DATA_DECRYPTED,
                          WRONG_KEY_ID, IMPROPER_DATA_LENGTH,
                          CRYPTO_PERIOD_EXPIRED,
                          CRYPTO_PERIOD_NOT_EXPIRED, BAD_ALGORITHM);
    icv_length_type = ...;
    mac_type = ...;
    mdc_type = ...;

(* Service Interaction Point - Type Definitions *)

channel CLNS_primitives (user, provider);
    (* This defines the interface between sp4e and its transport user,
       and also the interface between sp4e and its network service provider.
       The service defined is simply the connectionless unit data service.
    *)

    by user:
        NUDATAreq (source_address : N_address_type;
                   destination_address : N_address_type;
                   QOS_parameter_set : N_QOS_parameter_set_type;
                   NS_user_data : data_type);

    by provider:
        NUDATAind (source_address : N_address_type;
                   destination_address : N_address_type;
                   QOS_parameter_set : N_QOS_parameter_set_type;
                   NS_user_data : data_type);
```

```
channel KMGR_service (user, provider);
   (* This defines the interface between sp4e and the key manager.
      The services defined are:
         - to establish a cryptographic association for an NSAP pair
         - to replace an existing cryptographic association for an NSAP pair
         - to release a cryptographic association for an NSAP pair
         - to verify that a valid cryptographic association exists
   *)

   by user:
      ESTABreq (request_id : request_identifier_type;
            local_address : N_address_type;
            peer_address : N_address_type;
            qos : N_QOS_parameter_set_type);

      RELEASEreq (request_id : request_identifier_type;
             local_key_id : key_identifier_type);

      REPLACEreq (request_id : request_identifier_type;
             local_key_id : key_identifier_type);

      VERIFYreq (request_id : request_identifier_type;
            local_key_id : key_identifier_type;
            local_address : N_address_type;
            peer_address : N_address_type;
            qos : N_QOS_parameter_set_type);

   by provider:
      ESTABresp (request_id : request_identifier_type;
            local_key_id : key_identifier_type;
            peer_key_id : key_identifier_type;
            security_attributes : security_attributes_type;
            status : kmgr_response_type);

      REPLACEresp (request_id : request_identifier_type;
             local_key_id : key_identifier_type;
             peer_key_id : key_identifier_type;
             security_attributes : security_attributes_type;
             status : kmgr_response_type);

      VERIFYresp (request_id : request_identifier_type;
             peer_key_id : key_identifier_type;
             security_attributes : security_attributes_type;
             status : kmgr_response_type);
```

```
channel KMGT_transactions (initiator, responder);
    (*  This defines the interface between peer key manager entities.
        It defines a subset of the SDNS key management primitives which
        allow credentials to be exchanged, key attributes to be negotiated,
        and key replacement to occur.
    *)

    by initiator:
        EXCHCREDreq (request_id : request_identifier_type;
                     univ_id : key_identifier_type;
                     init_key_id : key_identifier_type;
                     init_credentials : credentials_type);

        ABORTreq (request_id : request_identifier_type;
                  init_key_id: key_identifier_type);

        SECATTRreq (request_id : request_identifier_type;
                    resp_key_id : key_identifier_type;
                    source_address : N_address_type;
                    destination_address : N_address_type;
                    proposed_options : security_attributes_type;
                    resp_obsolete_key_id : key_identifier_type);

        DELKEYreq (request_id : request_identifier_type;
                   init_key_id: key_identifier_type);

    by responder:
        EXCHCREDresp (request_id : request_identifier_type;
                      init_key_id : key_identifier_type;
                      resp_key_id : key_identifier_type;
                      resp_credentials : credentials_type;
                      result : kmgt_status_type);

        SECATTRresp (request_id : request_identifier_type;
                     init_key_id : key_identifier_type;
                     selected_options : security_attributes_type;
                     result : kmgt_status_type);
```

```
(* Module Headers *)


module  Transport_User process(role : role_type);

    ip  PseudoNSAP : array[NSAP_type] of CLNS_primitives (user);
        (* This interaction point is used by the transport user to exchange
           information with a peer.
        *)

export phase : phase_type;
end;


module  SP4E_Entity process(cf_id: cf_id_type);

    ip  PseudoNSAP: array[NSAP_type] of CLNS_primitives (provider);
        (* This interaction point models the pseudo connectionless network
           service access points offered to the transport user.
        *)

        NSAP: array[NSAP_type] of CLNS_primitives (user);
        (* This interaction point models the connectionless network service
           access points utilized by the sp4e entity.
        *)

        KEYservice: KMGR_service (user) common queue;
        (* This interaction point models the interface to the key manager
           utilized for establishing cryptographic associations.
        *)

end;


module  Comm_Network process;

    ip  NSAP: array[1..MAX_N_address] of CLNS_primitives (provider);
        (* This interaction point models the connectionless network service
           access points provided by the communications network.
        *)

end;
```

```
module  Key_Manager process(role: role_type; cf_id: cf_id_type);

    ip  KEYservice : KMGR_service (provider) common queue;
        (* This interaction point models the interface offered by the key
           manager to the sp4e entity.
        *)

        TOpeer : array[K_STREAM_type] of KMGT_transactions (initiator);
        (* This interaction point is used by the key manager to communicate
           directly with a peer key manager for simulation purposes.
        *)

        FROMpeer : array[K_STREAM_type] of KMGT_transactions (responder);
        (* This interaction point is used by the key manager to communicate
           directly with a peer key manager for simulation purposes.
        *)

export system_id : integer;
end;

(* Functions and Procedures *)

procedure init_cf(var cf_id: cf_id_type);
    (* Initialize the indicated cryptographic facility.
    *)
    primitive;

procedure shutdown_spec;
    (* Terminate the operation of this specification.
    *)
    primitive;

(* Bodies of Modules *)

body user_body for Transport_User;

type
    mode_type = (ping_pong, twa, tws);
    mmi_status_type = (no_input, input_ok, input_failed, input_terminated);

var
    mode : mode_type;
    mmi_status : mmi_status_type;
    outstanding_responses : integer;
```

```
    vpid : vpid_type;

(* Functions and Procedures *)

function determine_NSAP( address : N_address_type) : NSAP_type;
    (* This function maps network addresses to service
       access points.
    *)
    primitive;

(*  The following set of functions make up the initialization and display
    functions for the man-machine interface.  The indicated input functions
    are reserved exclusively for the transport user module.
*)

function event_available( vpid : vpid_type;
                          mode : mode_type;
                          mmi_status : mmi_status_type;
                          outstanding_responses : integer) : boolean;
    (* Determine whether a new transport event is available to be issued.
    *)
    primitive;

procedure read_next_event (vpid : vpid_type;
                           var source : N_address_type;
                           var destination : N_address_type;
                           var QOS : N_QOS_parameter_set_type;
                           var data : data_type;
                           var mmi_status : mmi_status_type);
    (* Get the next available transport event via the man-machine
       interface queue.
    *)
    primitive;

procedure write_last_event(vpid : vpid_type;
                           index : NSAP_type;
                           source : N_address_type;
                           destination : N_address_type;
                           QOS : N_QOS_parameter_set_type;
                           data : data_type);
    (* Display event information to the man-machine interface.
    *)
    primitive;
```

```
procedure open_viewport( role : role_type;
                         var vpid : vpid_type );
   (* Activate a man-machine interface window for this role.
   *)
   primitive;

(* end of mmi facility *)

state Active;

initialize
   to Active
      begin
         open_viewport (role, vpid);
         phase := start;
         outstanding_responses := 0;
         mmi_status := no_input;
         mode := ping_pong;
      end;

trans

from Active to Active
   provided event_available(vpid, mode, mmi_status, outstanding_responses)
         and (role = stimulate)
            (* In a stimulating role, the next user selected event must be
               issued whenever it becomes available.
            *)
            var
               index : NSAP_type;
               source : N_address_type;
               destination: N_address_type;
               QOS : N_QOS_parameter_set_type;
               data : data_type;
            begin
            read_next_event ( vpid, source, destination, QOS, data, mmi_status);
            if (mmi_status = input_ok) then
               begin
               index := determine_NSAP(destination);
               write_last_event( vpid, index, source, destination, QOS, data);
               output PseudoNSAP[index].NUDATAreq( source, destination,
                                                    QOS, data);
               outstanding_responses := outstanding_responses + 1;
               end
```

```
        else
            phase := finish;
        end;

from Active to Active
    any index: NSAP_type do
        when PseudoNSAP[index].NUDATAind
            provided (role = stimulate)
                (* In a stimulating role, report the event and update the
                   outstanding number of responder replies.
                *)
                begin
                write_last_event( vpid, index, source_address, destination_address,
                                  QOS_parameter_set, NS_user_data);
                outstanding_responses := outstanding_responses - 1;
                end;

            provided (role = respond)
                (* In a responding role, report the event and echo incoming
                   network data back to the stimulator.
                *)
                begin
                write_last_event( vpid, index, source_address, destination_address,
                                  QOS_parameter_set, NS_user_data);
                output PseudoNSAP[index].NUDATAreq(destination_address,
                             source_address, QOS_parameter_set, NS_user_data);
                write_last_event( vpid, index, destination_address, source_address,
                                  QOS_parameter_set, NS_user_data);
                end;

            provided (role = report)
                (* In a reporting role, merely report the event.
                *)
                begin
                write_last_event( vpid, index, source_address, destination_address,
                                  QOS_parameter_set, NS_user_data);
                end;

end;        { end of transport user module body }


body sp4e_body for SP4E_Entity;
external;   { see Appendix B }
```

```
body network_body for Comm_Network;

var
    vpid : vpid_type;

procedure write_last_event(vpid : vpid_type;
                           index : NSAP_type;
                           source : N_address_type;
                           destination : N_address_type;
                           QOS : N_QOS_parameter_set_type;
                           data : data_type);
    (* Display event information to the man-machine interface.
    *)
    primitive;

procedure open_viewport( role : role_type;
                         var vpid : vpid_type );
    (* Activate a man-machine interface window for this role.
    *)
    primitive;

state Active;

initialize
    to Active
        begin
        open_viewport (report, vpid);
        end;

trans
(* The following transition maps requests from NSAP i to indications
   on a complementary NSAP as computed by MAX_N_ADDRESS-i+1.
*)
    any index:1..MAX_N_ADDRESS do
        when NSAP[index].NUDATAreq
            from Active to Active
                begin
                output NSAP[MAX_N_ADDRESS-index+1].NUDATAind(source_address,
                            destination_address, QOS_parameter_set, NS_user_data);
                write_last_event( vpid, index, source_address, destination_address,
                                  QOS_parameter_set, NS_user_data);
                end;

end;        { end of network module body }
```

```
body kmgr_body for Key_Manager;
external;   { see Appendix C }

(* declaration and initialization of simulated environment *)

modvar
    User1, User2 : Transport_User;
    SP4E1, SP4E2 : SP4E_Entity;
    KMGR1, KMGR2 : Key_Manager;
    Network : Comm_Network;

var
    cf_id1, cf_id2: cf_id_type;

state Active;

initialize
    to Active
    { instantiate modules with appropriate body specifications }
    begin
    init_cf(cf_id1);
    init_cf(cf_id2);
    init User1 with user_body(stimulate);
    init User2 with user_body(respond);
    init KMGR1 with kmgr_body(stimulate, cf_id1);
    init SP4E1 with sp4e_body(cf_id1);
    init KMGR2 with kmgr_body(respond, cf_id2);
    init SP4E2 with sp4e_body(cf_id2);
    KMGR1.system_id := 1;
    KMGR2.system_id := 2;
    init Network with network_body;
    { connect modules together in desired configuration }
    all i : 1..MAX_K_STREAM do
        begin
        connect KMGR1.TOpeer[i] to KMGR2.FROMpeer[i];
        connect KMGR2.TOpeer[i] to KMGR1.FROMpeer[i];
        end;
    connect SP4E1.KEYservice to KMGR1.KEYservice;
    connect SP4E2.KEYservice to KMGR2.KEYservice;
    all i : 1..MAX_NSAP do
        begin
        connect User1.PseudoNSAP[i] to SP4E1.PseudoNSAP[i];
        connect User2.PseudoNSAP[i] to SP4E2.PseudoNSAP[i];
```

```
        connect SP4E1.NSAP[i] to Network.NSAP[i];
        connect SP4E2.NSAP[i] to Network.NSAP[MAX_N_ADDRESS-i+1];
        end;

    end;    { end of initialization }

trans

from Active to Active
    provided User1.phase = finish
begin
    all i : 1..MAX_NSAP do
        begin
        disconnect SP4E2.NSAP[i];
        disconnect SP4E1.NSAP[i];
        disconnect User2.PseudoNSAP[i];
        disconnect User1.PseudoNSAP[i];
        end;
    disconnect SP4E2.KEYservice;
    disconnect SP4E1.KEYservice;
    all i : 1..MAX_K_STREAM do
        begin
        disconnect KMGR2.TOpeer[i];
        disconnect KMGR1.TOpeer[i];
        end;
    release Network;
    release SP4E2;
    release KMGR2;
    release SP4E1;
    release KMGR1;
    release User2;
    release User1;
    shutdown_spec;
    User1.phase := shutdown;
end;

end.        { end of specification }
```

APPENDIX B
SPECIFICATION OF SP4E ENTITY SUBMODULES

This appendix contains the Estelle specification of the SP4E Entity submodules. The diagram below gives a general overview of the specification following the conventions of Appendix A.

```
+----SP4E Entity module body--------------------------
|
|      +----timer facility module header---------
|      |            T:
|      +------------------------------------
|
|      +----timer facility module body-----------
|
|      +----sp4e machine module header-------
|      |            P:  N:  K:  T:
|      +------------------------------------
|
|      +----sp4e machine module body----------
|
+---------------------------------------------
```

```
body sp4e_body for SP4E_Entity;

const
    my_identifier = sp4eentity;
    MAX_SP4_INSTANCES = any integer;

type
    sp4_occurrence_type = 1..MAX_SP4_INSTANCES;
    direction_type = (up_stack, down_stack);
    request_type = (none, kill_me);
    flow_control_type = (off, on);
    pdu_sequence_type = 0..2147483647;  { 0 to 2**31-1 }
    fsn_type = record
                    last_DT_sent : pdu_sequence_type;
                    last_DT_received : pdu_sequence_type;
                    last_ED_sent : pdu_sequence_type;
                    last_ED_received : pdu_sequence_type;
               end;
    timer_id_type = (key_manager, inactivity);
    severity_type = (INDETERMINATE, CRITICAL, MAJOR, MINOR);
    security_alarm_type = (INTEGRITY_VIOLATION, OPERATIONAL_VIOLATION,
                           PHYSICAL_VIOLATION,  SECURITY_DOMAIN_VIOLATION,
                            TIME_DOMAIN_VIOLATION);
    alarm_cause_type = (KEY_MANAGEMENT_FAILURE,
                         KEY_MANAGER_TIME_OUT,
                         UNENCAPSULATED_TPDU_RECEIVED,
                         ADDRESS_CHECK_FAILURE,
                        CRYPTO_FACILITY_FAILURE, INTEGRITY_CHECK_FAILURE,
                        MAX_SP4_INSTANCES_EXCEEDED, LABEL_CHECK_FAILURE,
                         REFLECTION_CHECK_FAILURE, FSN_FAILURE,
                         OTHER_FAILURE,
                         TRAFFIC_KEY_EXPIRED);

var
    sp4_free : array [1..MAX_SP4_INSTANCES] of boolean;

state    Active;
```

```
channel KMGR_primitives (user, provider);
   (* This defines the interface the between sp4_machine and the SP4E_Entity
      who multiplexes the transactions to the key manager.
      The services defined are:
         - to establish a cryptographic association for an NSAP pair
         - to replace a cryptographic association for an NSAP pair
         - to release a cryptographic association for an NSAP pair
         - to verify that a valid cryptographic association exists
   *)

   by user:
      ESTABreq (local_address : N_address_type;
                peer_address : N_address_type;
                qos : N_QOS_parameter_set_type);

      REPLACEreq (local_key_id : key_identifier_type);

      RELEASEreq (local_key_id : key_identifier_type);

      VERIFYreq (local_key_id : key_identifier_type;
                 local_address : N_address_type;
                 peer_address : N_address_type;
                 qos : N_QOS_parameter_set_type);

   by provider:
      ESTABresp (local_key_id : key_identifier_type;
                 peer_key_id : key_identifier_type;
                 security_attributes : security_attributes_type;
                 status : kmgr_response_type);

      REPLACEresp (local_key_id : key_identifier_type;
                   peer_key_id : key_identifier_type;
                   security_attributes : security_attributes_type;
                   status : kmgr_response_type);

      VERIFYresp (peer_key_id : key_identifier_type;
                  security_attributes : security_attributes_type;
                  status : kmgr_response_type);
```

```
channel TIMER_primitives(user, provider);
    (* This defines the interface between the sp4_machine and the timer
       facility.  The set event enables an individual timer, while the cancel
       event disables all timers.  The expiration of a timer is indicated
       with an alarm event.
    *)

    by user:
        start (timer_id : timer_id_type);

        cancel;

    by provider:
        alarm (timer_id : timer_id_type);

(* Functions and Procedures *)

procedure determine_label(qos: N_QOS_parameter_set_type;
                          var d_label: security_label_type);
     (* Determine the label conveyed or implied by the quality of service
        parameters and system environment.
     *)
    primitive;

function label_match( lbl: security_label_type;
                      qos: N_QOS_parameter_set_type) : boolean;
    (* Compare a security label with that which may be conveyed by a set of
       QOS parameters and return the status.
    *)
    primitive;

procedure report_security_event( identifier : name_type;
                                 severity : severity_type;
                                 class : security_alarm_type;
                                 reason : alarm_cause_type;
                                 local_address : N_address_type;
                                 peer_address : N_address_type);
    (* Report security errors to a system facility for management handling
       and processing.
    *)
    primitive;
```

```
function address_equal( a,b: N_address_type) : boolean;
    (* Compare two addresses and return the status.
    *)
    primitive;

function determine_origin( request_identifier : request_identifier_type) :
                              sp4_occurrence_type;
    (* Map a request identifier back to the sp4 machine that issued it.
    *)
    primitive;

function construct_id( index : sp4_occurrence_type) :
                              request_identifier_type;
    (* Construct a request identifier from an sp4 machine index.
    *)
    primitive;

(* SP4E Submodules *)

module timer_facility process;

    ip  T : TIMER_primitives (provider);

end;        { end of timer_facility module header }

body timer_body for timer_facility;

const
    MAX_KMGR_DELAY = any integer;
    MAX_INACT_DELAY = any integer;

var
    kmgr_timer_on : boolean;
    inact_timer_on : boolean;

state Active;

initialize
    to Active
    begin
    kmgr_timer_on := false;
    inact_timer_on := false;
    end;
```

```
trans

(* key manager timer processing *)

    from Active to Active
        provided kmgr_timer_on
            delay (MAX_KMGR_DELAY)
            begin
            kmgr_timer_on := false;
            output T.alarm( key_manager);
            end;

        when T.start
            provided (timer_id=key_manager)
            begin
            kmgr_timer_on := true;
            end;

(* inactivity timer processing *)

    from Active to Active
        provided inact_timer_on
            delay (MAX_INACT_DELAY)
            begin
            inact_timer_on := false;
            output T.alarm( inactivity);
            end;

        when T.start
            provided (timer_id=inactivity)
            begin
            inact_timer_on := true;
            end;

(* processing of all timers *)

    from Active to Active
        when T.cancel
            begin
            kmgr_timer_on := false;
            inact_timer_on := false;
            end;

end;        { end of timer body }
```

```
module sp4_machine process(cf_id: cf_id_type);

    ip  P : CLNS_primitives (provider);

        N : CLNS_primitives (user);

        K : KMGR_primitives (user);

        T : TIMER_primitives (user);

export  request : request_type;
        index : NSAP_type;
        hold_local_address : N_address_type;
        hold_peer_address : N_address_type;
        hold_label : security_label_type;
        fsn : fsn_type;
        flow_control : flow_control_type;

end;        { end of sp4_machine module header }

body sp4_body for sp4_machine;

const
    my_identifier = sp4machine;

type
    build_status_type = (BUILD_OK, MAC_ERROR, MDC_ERROR,
                         ENCIPHER_MECH_ERROR, KEY_EXPIRED);
    decode_status_type = (DECODE_OK, DECIPHER_MECH_ERROR,
                         INVALID_FLAG, INVALID_LABEL,
                         INVALID_FSN, INVALID_ICV, EXPIRED_KEY);
    TPDU_code_type = (CR, CC, DR, DC, DT, AK, ED, EA, RJ, ER, SE, invalid);
    sp4_occurrence_type = 1..MAX_SP4_INSTANCES;

var
    first_transition: boolean;
    direction : direction_type;
    key_replacement_required : boolean;

    (* hold area variables *)
    hold_local_key_id : key_identifier_type;
    hold_peer_key_id : key_identifier_type;
    hold_security_attributes : security_attributes_type;
```

```
    hold_QOS_parameter_set : N_QOS_parameter_set_type;
    hold_user_data : data_type;
    hold_encapsulated_data : data_type;
    hold_protected_data : data_type;

state IDLE, WFKE, WFKV, WFKR, ACTIVE;

(* Functions and Procedures *)

procedure init_label_table;
    (* Initialize tables associated with label processing.
    *)
    primitive;

procedure get_date(var today_date : date_type);
    (* Get today's date from the system.
    *)
    primitive;

function determine_kind(data : data_type) : TPDU_code_type;
    (* Determine the type of TPDU.
    *)
    primitive;

procedure report_security_event( identifier : name_type;
                                 severity : severity_type;
                                 class : security_alarm_type;
                                 reason : alarm_cause_type;
                                 local_address : N_address_type;
                                 peer_address : N_address_type);
    (* Report security errors that are encountered during the execution of
       the security protocol.
    *)
    primitive;

(* This set of functions is used by sp4 machine to encipher and
   decipher pdu information and generally speaking are independent of the
   cryptographic algorithms implemented.
*)

function CHECKtraffickey (cf_id: cf_id_type;
                          key_id: key_identifier_type) :
                          cipher_status_type ; primitive;
```

```
procedure COMPUTEmac ( cf_id: cf_id_type;
                    key_id : key_identifier_type;
                    integ_alg: algorithm_id_type;
                    data : data_type;
                    icv_len : icv_length_type;
                    var mac : mac_type;
                    var status : cipher_status_type); primitive;

procedure COMPUTEmdc ( cf_id: cf_id_type;
                    integ_alg: algorithm_id_type;
                    data : data_type;
                    icv_len : icv_length_type;
                    var mdc : mdc_type;
                    var status : cipher_status_type); primitive;

procedure ENCRYPT ( cf_id: cf_id_type;
                 key_id : key_identifier_type;
                 conf_alg: algorithm_id_type;
                 data : data_type;
                 var cipher_data : data_type;
                 var status : cipher_status_type); primitive;

procedure DECRYPT ( cf_id: cf_id_type;
                 key_id : key_identifier_type;
                 conf_alg: algorithm_id_type;
                 var data : data_type;
                 cipher_data : data_type;
                 var status : cipher_status_type); primitive;

(* end of cryptofacility definitions *)

procedure build_SE ( protected_data : data_type;
                  kind : TPDU_code_type;{}
                  fsn : fsn_type;
                  local_key_id : key_identifier_type;
                  peer_key_id : key_identifier_type;
                  security_attributes : security_attributes_type;
                  var encapsulated_data : data_type;
                  var build_status: build_status_type );
   (* Encapsulate "protected_data" applying the protection mechanisms
      indicated, and encode into an SE TPDU.
   *)
   primitive;
```

```
procedure decode_header ( user_data : data_type;
                          var kind : TPDU_code_type;
                          var local_key_id : key_identifier_type;
                          var encapsulated_data : data_type);
    (* Get header portion of TPDU and determine its type.
    *)
    primitive;

procedure decode_SE(encapsulated_data: data_type;
                    fsn: fsn_type;
                    local_key_id: key_identifier_type;
                    sec_att: security_attributes_type;
                    var protected_data: data_type;
                    var decode_status: decode_status_type);
    (* Decode the received SE TPDU, decapsulate the data, and verify
       that the indicated protection mechanisms are in effect.
    *)
    primitive;

initialize
    to IDLE
    begin
        init_label_table;
        first_transition := true;
        request := none;
        flow_control := off;
    end;
```

```
(* Transitions *)

trans

(* key establishment *)

  from IDLE
    to WFKE
      when P.NUDATAreq
        provided first_transition and (flow_control = off)
        (* Module instantiated through user event;  retain event
           information, and obtain a key and associated information
           from the key manager.
        *)
        begin
        first_transition := false;
        key_replacement_required := false;
        flow_control := on;
        hold_local_address := source_address;
        hold_peer_address := destination_address;
        hold_QOS_parameter_set := QOS_parameter_set;
        hold_user_data := NS_user_data;
        output T.start (key_manager);
        output K.ESTABreq (hold_local_address, hold_peer_address,
                          hold_QOS_parameter_set);
        end;

  from WFKE
    to IDLE
      when K.ESTABresp
        provided (status <> assoc_ok)
        (* Key manager responds negatively;  discard input, report
           errors, and request termination.
        *)
        begin
        output T.cancel;
        { discard input data }
        report_security_event( my_identifier, MAJOR,
                              SECURITY_DOMAIN_VIOLATION,
                              KEY_MANAGEMENT_FAILURE,
                              hold_local_address, hold_peer_address);
        flow_control := off;
        request := kill_me;
        end;
```

```
from WFKE
    to IDLE
        when T.alarm
            provided (timer_id = key_manager)
            (*  Key manager does not respond in time, discard input,
                report errors, and request termination.
            *)
            begin
            { discard input data }
            report_security_event( my_identifier, CRITICAL,
                                   TIME_DOMAIN_VIOLATION,
                                   KEY_MANAGER_TIME_OUT,
                                   hold_local_address, hold_peer_address);
            flow_control := off;
            request := kill_me;
            end;

from WFKE
    to ACTIVE
        when K.ESTABresp
            provided (status = assoc_ok)
                    { no key_replacement_required is assumed }
            (*  Key manager responds positively to an earlier request;
                save security association information, and protect user
                TPDU in accordance with negotiated security attributes.
            *)
            var encapsulated_data : data_type;
                build_status : build_status_type;
            begin
            output T.cancel;
            flow_control := off;
            hold_local_key_id := local_key_id;
            hold_peer_key_id := peer_key_id;
            hold_security_attributes := security_attributes;
            build_SE ( hold_user_data, determine_kind(hold_user_data), fsn,
                       local_key_id, peer_key_id, security_attributes,
                       encapsulated_data, build_status);
            if (build_status = BUILD_OK) then
                begin
                output T.start(inactivity);
                output N.NUDATAreq( hold_local_address, hold_peer_address,
                                    hold_QOS_parameter_set, encapsulated_data);
                end
```

```
            { otherwise discard problem tpdu }
            else if (build_status = MAC_ERROR) or
                  (build_status = MDC_ERROR) then
               report_security_event(my_identifier, MAJOR,
                                     INTEGRITY_VIOLATION,
                                     INTEGRITY_CHECK_FAILURE,
                                     hold_local_address, hold_peer_address)
            else if (build_status = ENCIPHER_MECH_ERROR) then
               report_security_event(my_identifier, MAJOR,
                                     SECURITY_DOMAIN_VIOLATION,
                                     CRYPTO_FACILITY_FAILURE,
                                     hold_local_address, hold_peer_address)
            else if (build_status = KEY_EXPIRED) then
               report_security_event(my_identifier, MAJOR,
                                     TIME_DOMAIN_VIOLATION,
                                     TRAFFIC_KEY_EXPIRED,
                                     hold_local_address, hold_peer_address)
            else
               report_security_event(my_identifier, MINOR,
                                     OPERATIONAL_VIOLATION,
                                     OTHER_FAILURE,
                                     hold_local_address, hold_peer_address);
            end;

(* key verification *)

   from IDLE
      to WFKV
         when N.NUDATAind
            provided first_transition and (flow_control = off)
            (* Module is instantiated through network event; must
               decode the header, check the type, and verify the key
               needed to unwrap the encapsulated data.
            *)
            var
               tpdu_id : TPDU_code_type;
            begin
            first_transition := false;
            key_replacement_required := false;
            flow_control := on;
            hold_local_address := destination_address;
            hold_peer_address := source_address;
            hold_QOS_parameter_set := QOS_parameter_set;
            hold_user_data := NS_user_data;
```

```
decode_header( hold_user_data, tpdu_id, hold_local_key_id,
               hold_encapsulated_data);
if (tpdu_id = SE) then
    begin
    output T.start( key_manager);
    output K.VERIFYreq( hold_local_key_id,
                        hold_local_address, hold_peer_address,
                        hold_QOS_parameter_set);
    end
else if (tpdu_id = invalid) then
    report_security_event( my_identifier, MINOR,
                           SECURITY_DOMAIN_VIOLATION,
                           OTHER_FAILURE,
                           hold_local_address, hold_peer_address)
else
    { discard problem tpdu }
    report_security_event( my_identifier, MAJOR,
                           SECURITY_DOMAIN_VIOLATION,
                           UNENCAPSULATED_TPDU_RECEIVED,
                           hold_local_address, hold_peer_address);
end;

from WFKV
    to IDLE
        when K.VERIFYresp
        provided (status <> assoc_ok)
        (*  Key manager unable to verify local key identifier
            conveyed by peer;  must discard TPDU and report
            security event.
        *)
        begin
        output T.cancel;
        { discard input data }
        report_security_event( my_identifier, MAJOR,
                               SECURITY_DOMAIN_VIOLATION,
                               KEY_MANAGEMENT_FAILURE,
                               hold_local_address, hold_peer_address);
        flow_control := off;
        request := kill_me;
        end;
```

```
from WFKV
    to IDLE
        when T.alarm
            provided (timer_id = key_manager)
            (*  Key manager unable to respond in time; must report security
                event and request termination.
            *)
            begin
            { discard input data }
            report_security_event( my_identifier, CRITICAL,
                                   TIME_DOMAIN_VIOLATION,
                                   KEY_MANAGER_TIME_OUT,
                                   hold_local_address, hold_peer_address);
            flow_control := off;
            request := kill_me;
            end;

from WFKV
    to ACTIVE
        when K.VERIFYresp
            provided (status = assoc_ok)
                    { no key_replacement_required is assumed }
            (*  Key manager responds with a valid peer key identifier;
                the incoming TPDU must be unwrapped, decoded, and
                output to the transport user.
            *)
            var
                protected_data : data_type;
                decode_status : decode_status_type;
            begin
            output T.cancel;
            flow_control := off;
            hold_security_attributes := security_attributes;
            hold_peer_key_id := peer_key_id;
            decode_SE ( hold_encapsulated_data, fsn,
                        hold_local_key_id, security_attributes,
                        protected_data, decode_status );
            if (decode_status = DECODE_OK) then
                begin
                output T.start(inactivity);
                output P.NUDATAind( hold_peer_address, hold_local_address,
                                    hold_QOS_parameter_set, protected_data);
                end
```

```
{ otherwise discard problem tpdu }
else if (decode_status = INVALID_ICV) then
    report_security_event(my_identifier, MAJOR,
                          INTEGRITY_VIOLATION,
                          INTEGRITY_CHECK_FAILURE,
                          hold_local_address, hold_peer_address)
else if (decode_status = INVALID_FLAG) then
    report_security_event(my_identifier, MAJOR,
                          INTEGRITY_VIOLATION,
                          REFLECTION_CHECK_FAILURE,
                          hold_local_address, hold_peer_address)
else if (decode_status = INVALID_FSN) then
    report_security_event(my_identifier, MINOR,
                          INTEGRITY_VIOLATION,
                          FSN_FAILURE,
                          hold_local_address, hold_peer_address)
else if (decode_status = INVALID_LABEL) then
    report_security_event(my_identifier, MAJOR,
                          SECURITY_DOMAIN_VIOLATION,
                          LABEL_CHECK_FAILURE,
                          hold_local_address, hold_peer_address)
else if (decode_status = DECIPHER_MECH_ERROR) then
    report_security_event(my_identifier, MAJOR,
                          SECURITY_DOMAIN_VIOLATION,
                          CRYPTO_FACILITY_FAILURE,
                          hold_local_address, hold_peer_address)
else if (decode_status = EXPIRED_KEY) then
    report_security_event(my_identifier, MAJOR,
                          TIME_DOMAIN_VIOLATION,
                          TRAFFIC_KEY_EXPIRED,
                          hold_local_address, hold_peer_address)
else
    report_security_event(my_identifier, MINOR,
                          OPERATIONAL_VIOLATION,
                          OTHER_FAILURE,
                          hold_local_address, hold_peer_address);
end;
```

```
(* encapsulation *)

    from ACTIVE
        to ACTIVE
            when N.NUDATAind
                provided (not key_replacement_required) and (flow_control = off)
                (*  The TPDU received from network must be unwrapped, decoded,
                    and output to the transport user.
                *)
                var
                    tpdu_id : TPDU_code_type;
                    protected_data : data_type;
                    local_key_id : key_identifier_type;
                    decode_status : decode_status_type;
                begin
                output T.cancel;
                decode_header( NS_user_data, tpdu_id, local_key_id,
                               hold_encapsulated_data);
                if (tpdu_id <> SE) then
                    begin
                    if (tpdu_id = invalid) then
                        report_security_event( my_identifier, MINOR,
                                               SECURITY_DOMAIN_VIOLATION,
                                               OTHER_FAILURE,
                                               hold_local_address, hold_peer_address)
                    else
                        report_security_event( my_identifier, MAJOR,
                                              SECURITY_DOMAIN_VIOLATION,
                                              UNENCAPSULATED_TPDU_RECEIVED,
                                              hold_local_address, hold_peer_address);
                    end
                { otherwise discard problem tpdu, unless key expired }
                else if (local_key_id <> hold_local_key_id) then
                    if (CHECKtraffickey( cf_id, hold_local_key_id) =
                        CRYPTO_PERIOD_EXPIRED) then
                        begin
                        flow_control := on;
                        { trigger spontaneous replacement event }
                        key_replacement_required := true;
                        hold_user_data := NS_user_data;
                        hold_QOS_parameter_set := QOS_parameter_set;
                        direction := up_stack;
                        end
```

```
        else
            begin
            report_security_event( my_identifier, MAJOR,
                                   SECURITY_DOMAIN_VIOLATION,
                                   ADDRESS_CHECK_FAILURE,
                                   hold_local_address, hold_peer_address);
            end
    else
        begin
        decode_SE ( hold_encapsulated_data, fsn, hold_local_key_id,
                hold_security_attributes, protected_data, decode_status );
        if (decode_status = DECODE_OK) then
            begin
            output T.start(inactivity);
            output P.NUDATAind( hold_peer_address, hold_local_address,
                                QOS_parameter_set, protected_data);
            end
        { otherwise discard problem tpdu, unless key expired }
        else if (decode_status = INVALID_ICV) then
            report_security_event(my_identifier, MAJOR,
                                  INTEGRITY_VIOLATION,
                                  INTEGRITY_CHECK_FAILURE,
                                  hold_local_address, hold_peer_address)
        else if (decode_status = INVALID_FLAG) then
            report_security_event(my_identifier, MAJOR,
                                  INTEGRITY_VIOLATION,
                                  REFLECTION_CHECK_FAILURE,
                                  hold_local_address, hold_peer_address)
        else if (decode_status = INVALID_FSN) then
            report_security_event(my_identifier, MINOR,
                                  INTEGRITY_VIOLATION,
                                  FSN_FAILURE,
                                  hold_local_address, hold_peer_address)
        else if (decode_status = INVALID_LABEL) then
            report_security_event(my_identifier, MAJOR,
                                  SECURITY_DOMAIN_VIOLATION,
                                  LABEL_CHECK_FAILURE,
                                  hold_local_address, hold_peer_address)
        else if (decode_status = DECIPHER_MECH_ERROR) then
            report_security_event(my_identifier, MAJOR,
                                  SECURITY_DOMAIN_VIOLATION,
                                  CRYPTO_FACILITY_FAILURE,
                                  hold_local_address, hold_peer_address)
```

```
        else if (decode_status = EXPIRED_KEY) then
            begin
            report_security_event(my_identifier, MINOR,
                                  TIME_DOMAIN_VIOLATION,
                                  TRAFFIC_KEY_EXPIRED,
                                  hold_local_address, hold_peer_address);
            flow_control := on;
            { trigger spontaneous replacement event}
            key_replacement_required := true;
            hold_user_data := NS_user_data;
            hold_QOS_parameter_set := QOS_parameter_set;
            direction := up_stack;
            end
        else
            report_security_event(my_identifier, MINOR,
                                  OPERATIONAL_VIOLATION,
                                  OTHER_FAILURE,
                                  hold_local_address, hold_peer_address);
        end;
    end;

from ACTIVE
    to ACTIVE
        when P.NUDATAreq
            provided (not key_replacement_required) and (flow_control = off)

            (*  TSDU received from user must be protected according
                to negotiated security mechanisms.
            *)
            var
                encapsulated_data : data_type;
                build_status : build_status_type;
            begin
            output T.cancel;
            build_SE ( NS_user_data, determine_kind(NS_user_data), fsn,
                    hold_local_key_id, hold_peer_key_id, hold_security_attributes,
                    encapsulated_data, build_status);
            if (build_status = BUILD_OK) then
                begin
                output T.start(inactivity);
                output N.NUDATAreq( hold_local_address, hold_peer_address,
                                QOS_parameter_set, encapsulated_data);
                end
```

```
{ otherwise discard problem tpdu, unless key expired }
else if (build_status = MAC_ERROR) or
        (build_status = MDC_ERROR) then
    report_security_event(my_identifier, MAJOR,
                          INTEGRITY_VIOLATION,
                          INTEGRITY_CHECK_FAILURE,
                          hold_local_address, hold_peer_address)
else if (build_status = ENCIPHER_MECH_ERROR) then
    report_security_event(my_identifier, MAJOR,
                          SECURITY_DOMAIN_VIOLATION,
                          CRYPTO_FACILITY_FAILURE,
                          hold_local_address, hold_peer_address)
else if (build_status = KEY_EXPIRED) then
    begin
    report_security_event(my_identifier, MINOR,
                          TIME_DOMAIN_VIOLATION,
                          TRAFFIC_KEY_EXPIRED,
                          hold_local_address, hold_peer_address);
    flow_control := on;
    { trigger spontaneous replacement event }
    key_replacement_required := true;
    hold_user_data := NS_user_data;
    hold_QOS_parameter_set := QOS_parameter_set;
    direction := down_stack;
    end
else
    report_security_event(my_identifier, MINOR,
                          OPERATIONAL_VIOLATION,
                          OTHER_FAILURE,
                          hold_local_address, hold_peer_address);
end;

(* key replacement *)

    from ACTIVE
        to WFKR
            provided key_replacement_required
                (*  Key replacement necessary;  invoke key manager to
                    update key.
                *)
                begin
                output T.start (key_manager);
                output K.REPLACEreq (hold_local_key_id);
                end;
```

```
from WFKR
    to IDLE
        when K.REPLACEresp
            provided (status <> assoc_ok)
            (* Key manager unable to replace key; must report event and
               request termination.
            *)
            begin
            output T.cancel;
            { discard input data }
            report_security_event( my_identifier, MAJOR,
                                   SECURITY_DOMAIN_VIOLATION,
                                   KEY_MANAGEMENT_FAILURE,
                                   hold_local_address, hold_peer_address);
            flow_control := off;
            request := kill_me;
            end;

from WFKR
    to IDLE
        when T.alarm
            provided (timer_id=key_manager)
            (*  Key manager unable to respond in time; must report security
                event and request termination.
            *)
            begin
            { discard input data }
            report_security_event( my_identifier, CRITICAL,
                                   TIME_DOMAIN_VIOLATION,
                                   KEY_MANAGER_TIME_OUT,
                                   hold_local_address, hold_peer_address);
            flow_control := off;
            request := kill_me;
            end;
```

```
from WFKR
    to ACTIVE
        when K.REPLACEresp
            provided (direction=down_stack) and (status=assoc_ok)
            (*  Key replacement complete; continue building SE TPDU for
                output to network.
            *)
            var
                encapsulated_data : data_type;
                build_status : build_status_type;
            begin
            output T.cancel;
            key_replacement_required := false;
            flow_control := off;
            hold_local_key_id := local_key_id;
            hold_peer_key_id := peer_key_id;
            hold_security_attributes := security_attributes;
            build_SE ( hold_user_data, determine_kind(hold_user_data), fsn,
                       hold_local_key_id, hold_peer_key_id, hold_security_attributes,
                       encapsulated_data, build_status);
            if (build_status = BUILD_OK) then
                begin
                output T.start(inactivity);
                output N.NUDATAreq( hold_local_address, hold_peer_address,
                                    hold_QOS_parameter_set, encapsulated_data);
                end
            { otherwise discard problem tpdu }
            else if (build_status = MAC_ERROR) or
                    (build_status = MDC_ERROR) then
                report_security_event(my_identifier, MAJOR,
                                      INTEGRITY_VIOLATION,
                                      INTEGRITY_CHECK_FAILURE,
                                      hold_local_address, hold_peer_address)
            else if (build_status = ENCIPHER_MECH_ERROR) then
                report_security_event(my_identifier, MAJOR,
                                      SECURITY_DOMAIN_VIOLATION,
                                      CRYPTO_FACILITY_FAILURE,
                                      hold_local_address, hold_peer_address)
            else if (build_status = KEY_EXPIRED) then
                report_security_event(my_identifier, MAJOR,
                                      SECURITY_DOMAIN_VIOLATION,
                                      KEY_MANAGEMENT_FAILURE,
                                      hold_local_address, hold_peer_address)
            else
```

```
            report_security_event(my_identifier, MINOR,
                                  OPERATIONAL_VIOLATION,
                                  OTHER_FAILURE,
                                  hold_local_address, hold_peer_address);
        end;

from WFKR
    to ACTIVE
        when K.REPLACEresp
            provided (direction=up_stack) and (status=assoc_ok)
            (*  Key replacement complete; continue decoding/decapsulation
                of SE TPDU for output to user.
            *)
            var
                protected_data : data_type;
                tpdu_id : TPDU_code_type;
                decode_status : decode_status_type;
            begin
            output T.cancel;
            key_replacement_required := false;
            flow_control := off;
            hold_local_key_id := local_key_id;
            hold_peer_key_id := peer_key_id;
            hold_security_attributes := security_attributes;
            decode_header(hold_user_data, tpdu_id, local_key_id,
                          hold_encapsulated_data);
            { already know tpdu is SE }
            if (local_key_id <> hold_local_key_id) then
                begin
                report_security_event( my_identifier, MAJOR,
                                      SECURITY_DOMAIN_VIOLATION,
                                      ADDRESS_CHECK_FAILURE,
                                      hold_local_address, hold_peer_address);
                end
            else
                begin
                decode_SE ( hold_encapsulated_data, fsn, hold_local_key_id,
                            hold_security_attributes, protected_data, decode_status );
                if (decode_status = DECODE_OK) then
                    begin
                    output T.start(inactivity);
                    output P.NUDATAind( hold_peer_address, hold_local_address,
                                        hold_QOS_parameter_set, protected_data);
                    end
```

```
    { otherwise discard problem tpdu }
    else if (decode_status = INVALID_ICV) then
        report_security_event(my_identifier, MAJOR,
                              INTEGRITY_VIOLATION,
                              INTEGRITY_CHECK_FAILURE,
                              hold_local_address, hold_peer_address)
    else if (decode_status = INVALID_FLAG) then
        report_security_event(my_identifier, MAJOR,
                              INTEGRITY_VIOLATION,
                              REFLECTION_CHECK_FAILURE,
                              hold_local_address, hold_peer_address)
    else if (decode_status = INVALID_FSN) then
        report_security_event(my_identifier, MINOR,
                              INTEGRITY_VIOLATION,
                              FSN_FAILURE,
                              hold_local_address, hold_peer_address)
    else if (decode_status = INVALID_LABEL) then
        report_security_event(my_identifier, MAJOR,
                              SECURITY_DOMAIN_VIOLATION,
                              LABEL_CHECK_FAILURE,
                              hold_local_address, hold_peer_address)
    else if (decode_status = DECIPHER_MECH_ERROR)then
        report_security_event(my_identifier, MAJOR,
                              SECURITY_DOMAIN_VIOLATION,
                              CRYPTO_FACILITY_FAILURE,
                              hold_local_address, hold_peer_address)
    else if (decode_status = EXPIRED_KEY) then
        report_security_event(my_identifier, MAJOR,
                              TIME_DOMAIN_VIOLATION,
                              TRAFFIC_KEY_EXPIRED,
                              hold_local_address, hold_peer_address)
    else
        report_security_event(my_identifier, MINOR,
                              OPERATIONAL_VIOLATION,
                              OTHER_FAILURE,
                              hold_local_address, hold_peer_address);
    end;
end;
```

```
(* inactivity *)

    from ACTIVE
        to IDLE
            when T.alarm
                provided (timer_id=inactivity)
                (*  No activity for a long time; must release key and
                    request termination.
                *)
                begin
                output K.RELEASEreq( hold_local_key_id);
                request := kill_me;
                end;

end;        { end of sp4m body }


    ip  P : array [sp4_occurrence_type] of CLNS_primitives (user);

        N : array [sp4_occurrence_type] of CLNS_primitives (provider);

        K : array [sp4_occurrence_type] of KMGR_primitives (provider);

modvar
    SP4 : array [sp4_occurrence_type] of sp4_machine;
    TIMER : array [sp4_occurrence_type] of timer_facility;

initialize
    to Active
        begin
        all i:sp4_occurrence_type do sp4_free[i] := true;
        end;
```

```
(* Transitions *)
trans

(* pseudo NSAP and NSAP multiplexing *)

from Active to Active
    any index: NSAP_type do
        when PseudoNSAP[index].NUDATAreq
            (* Instantiate an sp4 machine for the initial incoming request on
               a pseudo network service access point, or if one is active,
               output the request to it.
            *)
            begin
            forone i: sp4_occurrence_type suchthat (not sp4_free[i]) and
                address_equal (SP4[i].hold_local_address, source_address) and
                address_equal (SP4[i].hold_peer_address, destination_address)
                and label_match (SP4[i].hold_label, QOS_parameter_set) do
                output P[i].NUDATAreq(source_address, destination_address,
                                      QOS_parameter_set, NS_user_data)
            otherwise
                begin
                forone i: sp4_occurrence_type suchthat sp4_free[i] do
                    begin
                    sp4_free[i] := false;
                    init TIMER[i] with timer_body;
                    init SP4[i] with sp4_body(cf_id);
                    SP4[i].index := index;
                    determine_label(QOS_parameter_set, SP4[i].hold_label);
                    connect P[i] to SP4[i].P;
                    connect N[i] to SP4[i].N;
                    connect K[i] to SP4[i].K;
                    connect TIMER[i].T to SP4[i].T;
                    output P[i].NUDATAreq(source_address, destination_address,
                                          QOS_parameter_set, NS_user_data)
                    end
                otherwise
                    begin
                    report_security_event( my_identifier, MAJOR,
                                           OPERATIONAL_VIOLATION,
                                           MAX_SP4_INSTANCES_EXCEEDED,
                                           source_address, destination_address);
                    end;
                end;
            end;
```

```
any index: NSAP_type do
    when NSAP[index].NUDATAind
        (* Instantiate an sp4 machine for the initial incoming indication
           on a network service access point, or if one is active,
           output the indication to it.
        *)
        begin
        forone i:sp4_occurrence_type suchthat (not sp4_free[i]) and
            address_equal(SP4[i].hold_local_address, destination_address)
            and address_equal(SP4[i].hold_peer_address, source_address)
            and label_match (SP4[i].hold_label, QOS_parameter_set) do
            output N[i].NUDATAind(source_address, destination_address,
                                   QOS_parameter_set, NS_user_data)
        otherwise
            begin
            forone i:sp4_occurrence_type suchthat sp4_free[i] do
                begin
                sp4_free[i] := false;
                init TIMER[i] with timer_body;
                init SP4[i] with sp4_body(cf_id);
                SP4[i].index := index;
                determine_label(QOS_parameter_set, SP4[i].hold_label);
                connect P[i] to SP4[i].P;
                connect N[i] to SP4[i].N;
                connect K[i] to SP4[i].K;
                connect TIMER[i].T to SP4[i].T;
                output N[i].NUDATAind(source_address, destination_address,
                                       QOS_parameter_set, NS_user_data)
                end
            otherwise
                begin
                report_security_event( my_identifier, MAJOR,
                                       OPERATIONAL_VIOLATION,
                                       MAX_SP4_INSTANCES_EXCEEDED,
                                       destination_address, source_address);
                end;
            end;
        end;
```

```
any i : sp4_occurrence_type do
    when P[i].NUDATAind
        begin
        output PseudoNSAP [SP4[i].index].NUDATAind(source_address,
                                                   destination_address,
                                                   QOS_parameter_set,
                                                   NS_user_data);
        end;

any i : sp4_occurrence_type do
    when N[i].NUDATAreq
        begin
        output NSAP [SP4[i].index].NUDATAreq(source_address,
                                             destination_address,
                                             QOS_parameter_set, NS_user_data);
        end;

(* key manager multiplexing *)

any i : sp4_occurrence_type do
    when K[i].ESTABreq
        begin
        output KEYservice.ESTABreq(construct_id(i), local_address,
                                   peer_address, qos);
        end;

any i : sp4_occurrence_type do
    when K[i].VERIFYreq
        begin
        output KEYservice.VERIFYreq(construct_id(i), local_key_id,
                                    local_address, peer_address, qos);
        end;

any i : sp4_occurrence_type do
    when K[i].RELEASEreq
        begin
        output KEYservice.RELEASEreq(construct_id(i), local_key_id);
        end;

any i : sp4_occurrence_type do
    when K[i].REPLACEreq
        begin
        output KEYservice.REPLACEreq(construct_id(i), local_key_id);
        end;
```

```
trans

    when KEYservice.ESTABresp
            var
                i : sp4_occurrence_type;
            begin
            i := determine_origin(request_id);
            output K[i].ESTABresp(local_key_id, peer_key_id,
                                  security_attributes, status);
            end;

    when KEYservice.VERIFYresp
            var
                i : sp4_occurrence_type;
            begin
            i := determine_origin(request_id);
            output K[i].VERIFYresp(peer_key_id, security_attributes, status);
            end;

    when KEYservice.REPLACEresp
            var
                i : sp4_occurrence_type;
            begin
            i := determine_origin(request_id);
            output K[i].REPLACEresp(local_key_id, peer_key_id,
                                    security_attributes, status);
            end;
```

```
(* termination request processing *)

trans

any i : sp4_occurrence_type do  {translator dependent code}{}
    provided (not sp4_free[i]) and (SP4[i].request=kill_me)
        from Active to Active
            (* Release any sp4e machine that requests termination, along
               with its associated timer_facility.
            *)
            begin
            disconnect SP4[i].T;
            disconnect P[i];
            disconnect N[i];
            disconnect K[i];
            release TIMER[i];
            release SP4[i];
            sp4_free[i] := true;
            end;

end;         { end of sp4e module body }
```

APPENDIX C
SPECIFICATION OF KEY MANAGER SUBMODULES

This appendix contains the Estelle specification of the Key Manager entity submodules. The diagram below gives a general overview of the specification following the conventions of Appendix A.

```
+----Key Manager module body-----------------------
|
|       +----KM Initiator module header-------
|       |             KEY:  I2R:
|       +-----------------------------------
|
|       +----KM Initiator module body----------
|
|       +----KM Responder module header---
|       |             R2I:
|       +-----------------------------------
|
|       +----KM Responder module body------
+-------------------------------------------
```

```
body kmgr_body for Key_Manager;

const
    my_identifier = keymanager;
    MAX_KM_INSTANCES = any integer;
    null_key_id = any key_identifier_type;
    Null_Address = any N_address_type;

type
    km_occurrence_type = 1..MAX_KM_INSTANCES;
    mode_type = (INITIATOR, RESPONDER);
    severity_type = (INDETERMINATE, CRITICAL, MAJOR, MINOR);
    alarm_type = (INTEGRITY_VIOLATION, OPERATIONAL_VIOLATION,
                  PHYSICAL_VIOLATION, SECURITY_DOMAIN_VIOLATION,
                  TIME_DOMAIN_VIOLATION);
    alarm_cause_type = (UNSUPPORTED_FEATURE, AUTHORIZATION_FAILURE,
                        CERTIFICATE_RETRIEVAL_FAILURE,
                        ASSOCIATION_SEEK_FAILURE,
                        ASSOCIATION_RETRIEVAL_FAILURE,
                        RESTRICTION_RETRIEVAL_FAILURE,
                        KEY_RETRIEVAL_FAILURE,
                        MAX_KM_INSTANCES_EXCEEDED,
                        KM_INITIATOR_NOT_FOUND,
                        KM_RESPONDER_NOT_FOUND,
                        CRYPTO_FACILITY_FAILURE);

    (* access control related *)
    access_control_status_type = (AUTH_OK, AUTH_NOT_OK);

    (* smib related *)
    association_identifier_type = ...;
    association_type = (EXT_ESTAB, INT_ESTAB);
    association_status_type = (INCOMPLETE, ESTABLISHED, NOT_VALID);
    initiated_by_type = (EITHER, LOCAL, PEER);
    association_attributes_type = record
            full_data: boolean;
            type_of_assoc: association_type;
            status: association_status_type;
            local_address: N_address_type;
            peer_address: N_address_type;
            initiated_by: initiated_by_type;
            security_id: smib_id_type;
            local_credentials: credentials_type;
            peer_credentials: credentials_type;
```

```
            cred_encrypting_key: key_identifier_type;
            smib_key_id: smib_id_type;
            local_key_id: key_identifier_type;
            peer_key_id: key_identifier_type;
                                 end;
    smib_status_type = (CANT_FIND_PA_ATT, CANT_FIND_C_ATT,
                    CANT_FIND_AA_ATT, CANT_FIND_R_ATT,
                    CANT_FIND_K_ATT, FOUND_CERT,
                    FOUND_PA_ATT, FOUND_C_ATT, FOUND_AA_ATT,
                    FOUND_R_ATT, FOUND_K_ATT,
                    DEL_PA_ATT, DEL_C_ATT, DEL_AA_ATT,
                    DEL_R_ATT, DEL_K_ATT, NEW_PA_ATT, NEW_C_ATT,
                    NEW_AA_ATT, NEW_R_ATT, NEW_K_ATT);

var
    my_certificate : certificate_type;
    smib_status : smib_status_type;
    crypto_status : crypto_status_type;
    kmi_free:array[km_occurrence_type] of boolean;
    kmr_free:array[km_occurrence_type] of boolean;

channel KMGR_primitives (user, provider);
    (* This defines the interface between the key manager initiator and the
       key manager who multiplexes transactions to/from the SP4 Entity.
       The services defined are:
           - to establish a cryptographic association for an NSAP pair
           - to replace a cryptographic association for an NSAP pair
           - to release a cryptographic association for an NSAP pair
           - to verify that a valid cryptographic association exists
    *)

    by user:
        ESTABreq (local_address : N_address_type;
                  peer_address : N_address_type;
                  qos : N_QOS_parameter_set_type);

        REPLACEreq (local_key_id : key_identifier_type);

        RELEASEreq (local_key_id : key_identifier_type);

        VERIFYreq (local_key_id : key_identifier_type;
                   local_address : N_address_type;
                   peer_address : N_address_type;
                   qos : N_QOS_parameter_set_type);
```

by provider:

```
ESTABresp (local_key_id : key_identifier_type;
           peer_key_id : key_identifier_type;
           security_attributes : security_attributes_type;
           status : kmgr_response_type);

REPLACEresp (local_key_id : key_identifier_type;
             peer_key_id : key_identifier_type;
             security_attributes : security_attributes_type;
             status : kmgr_response_type);

VERIFYresp (peer_key_id : key_identifier_type;
            security_attributes : security_attributes_type;
            status : kmgr_response_type);

RELEASEresp; { used to determine when to terminate KM initiator }
```

```
channel KMGT_primitives (initiator, responder);
    (* This defines the interface between both the key manager initiator and
       responder entities and the key manager who multiplexes transactions
       to/from them. It defines a subset of the SDNS key management
       primitives that allow credentials to be exchanged, and key attributes
       to be negotiated. It also defines additional primitives needed for
       control of submodules.
    *)

    by initiator:
        EXCHCREDreq (univ_id : key_identifier_type;
                     init_key_id : key_identifier_type;
                     init_credentials : credentials_type);

        SECATTRreq (resp_key_id : key_identifier_type;
                    source_address : N_address_type;
                    destination_address : N_address_type;
                    proposed_options : security_attributes_type;
                    resp_obsolete_key_id : key_identifier_type);

        ABORTreq (init_key_id :key_identifier_type);
                 { added to simulate A_abort ACSE }

        DELKEYreq (init_key_id : key_identifier_type);
                  { added to prevent buildup of unwanted keys }
```

```
  by responder:
     EXCHCREDresp (init_key_id : key_identifier_type;
                   resp_key_id : key_identifier_type;
                   resp_credentials : credentials_type;
                   result : kmgt_status_type);

     SECATTRresp (init_key_id : key_identifier_type;
                  selected_options : security_attributes_type;
                  result : kmgt_status_type);

     ABORTresp; { used to determine when to terminate KM responder }

     DELKEYresp; { used to determine when to terminate KM responder }

state Active;

(* Functions and Procedures *)

procedure init_smib(role: role_type);
  (* Initialize the security management information base for this role.
  *)
  primitive;

procedure GET_CERT ( role: role_type;
                     var certificate : certificate_type;
                     var status : smib_status_type);
  (* This routine retrieves the system certificate from the Smib.
  *)
  primitive;

procedure load_cf( role: role_type;
                   cf_id: cf_id_type);
  (* Used to initialize the cryptofacility with every key in the Smib that
     represents an externally established active association.
  *)
  primitive;

function determine_stream( request_identifier : request_identifier_type) :
                           K_STREAM_type;
  (* Maps a request identifier into a particular key management stream.
  *)
  primitive;
```

```
(* The following set of functions and procedures represent the part of the
   cryptographic facility used by the key manager to administer key
   material within the facility, and is therefore dependent upon the key
   management scheme implemented. The functions below are for a credentials
   based scheme.
*)

function LOADcertificate ( cf_id: cf_id_type;
                           certificate : certificate_type;
                           universal_id : key_identifier_type) :
                           crypto_status_type; primitive;
    (* Used to load a system certificate into the cryptographic facility.
    *)

procedure LOADtraffickey ( cf_id: cf_id_type;
                           universal_id : key_identifier_type;
                           key_id : key_identifier_type;
                           key_attributes : key_attributes_type;
                           var status :crypto_status_type); primitive;
    (* Used to load a traffic key into the cryptofacility, either for
       manually distributed keys or staged keys.
    *)

procedure BEGINtraffickey ( cf_id: cf_id_type;
                            universal_id : key_identifier_type;
                            var local_credentials : credentials_type;
                            var key_id : key_identifier_type;
                            var status : crypto_status_type); primitive;
     (* Used in conjunction with the COMPLETEtraffickey function by the
        initiating side of a cryptographic association to generate a
        traffic key based on the exchange of credentials.
     *)

procedure COMPLETEtraffickey ( cf_id: cf_id_type;
                               universal_id : key_identifier_type;
                               peer_credentials : credentials_type;
                               key_id : key_identifier_type;
                               var key_attributes : key_attributes_type;
                               status : crypto_status_type); primitive;
     (* Used to complete the generation of a traffic key begun with the
        BEGINtraffickey function, once the peer's credentials have been
        received.
     *)
```

```
procedure GENERATEtraffickey ( cf_id: cf_id_type;
                               universal_id : key_identifier_type;
                               peer_credentials : credentials_type;
                               var local_credentials : credentials_type;
                               var key_id : key_identifier_type;
                               var key_attributes : key_attributes_type;
                               var status :crypto_status_type); primitive;
        (* Used by the responding side of a cryptographic association to
           generate a traffic key based on the receipt of credentials.
        *)

function RELEASEtraffickey ( cf_id: cf_id_type;
                             key_id : key_identifier_type) :
                             crypto_status_type; primitive;
        (* Used to release the local storage of a key whose cryptographic
           association has been terminated.
        *)

(* end of cryptofacility definitions *)

procedure report_event( identifier : name_type;
                        severity : severity_type;
                        class : alarm_type;
                        reason : alarm_cause_type;
                        local_address : N_address_type;
                        peer_address : N_address_type);
    (* Used to notify systems management of any security errors encountered
       during key management processing.
    *)
    primitive;
```

```
module  KM_initiator process(role: role_type; cf_id: cf_id_type);

    ip  K : KMGR_primitives (provider) common queue;
        (* This interaction point models the interface offered by the key
           manager to the sp4e entity.
        *)

        I2R : KMGT_primitives (initiator);
        (* This interaction point is used by an initiator submodule of the key
           manager to communicate directly with the responder side which lies
           within the peer key manager.
        *)

export  id : request_identifier_type;
end;

body kmi_body for KM_initiator;

const
    my_identifier = kminitiator;
    null_key_attributes = any key_attributes_type;
    null_security_attributes = any security_attributes_type;
    null_address = any N_address_type;

var
    assoc_attributes : association_attributes_type;
    hold_security_attributes : security_attributes_type;
    hold_key_attributes : key_attributes_type;
    source : N_address_type;
    destination : N_address_type;
    smib_index: smib_id_type;

state IDLE, WFPA, WFAC;

(* Functions and Procedures *)

function AUTHassoc(role: role_type;
                   smib_id: smib_id_type;
                   mode:mode_type):access_control_status_type;
    (* Given a smib index, this routine checks to see if the corresponding
       Permitted Association(s) constraints are met.
    *)
    primitive;
```

```
function available_SA_in_SMIB(role: role_type;
                              local_address:N_address_type;
                              peer_address:N_address_type;
                              ps_qos: N_QOS_value_type): boolean;
    (* Given two addresses and protection QOS, this routine looks for the
       corresponding Active Association(s) and, if one is found, checks to see
       if it references an available set of restraints.
    *)
    primitive;

procedure ADDRESS_SEEK_PA(role: role_type;
                         local_address: N_address_type;
                         peer_address: N_address_type;
                         ps_qos: N_QOS_value_type;
                         var smib_id: smib_id_type;
                         var status: smib_status_type);
    (* Given two addresses and protection QOS, this routine returns the index of
       the Permitted Association(s) with the matching addresses.
    *)
    primitive;

procedure ADDRESS_SEEK_AA(role: role_type;
                         local_address: N_address_type;
                         peer_address: N_address_type;
                         ps_qos: N_QOS_value_type;
                         var smib_id: smib_id_type;
                         var status: smib_status_type);
    (* Given two addresses and protection QOS, this routine returns the index of
       the Active Association(s) with the matching addresses.
    *)
    primitive;

procedure AA_info_replace(role: role_type;
                         smib_id: smib_id_type;
                         assoc_attributes:association_attributes_type;
                         security_attributes:security_attributes_type;
                         key_attributes:key_attributes_type;
                         var replacement_status:smib_status_type);
    (* Given a smib index, this routine replaces the corresponding Active
       Association(s) and the security and key attributes that it refers to
        with the ones given.
    *)
    primitive;
```

```
procedure AA_info_retrieve(role: role_type;
                           smib_id: smib_id_type;
                           var assoc_attributes:association_attributes_type;
                           var security_attributes:security_attributes_type;
                           var key_attributes:key_attributes_type;
                           var retrieval_status:smib_status_type);
    (* Given a smib index, this routine retrieves the corresponding Active
       Association(s) and the security and key attributes that it refers to.
    *)
    primitive;

procedure AA_info_delete( role: role_type;
                          smib_id: smib_id_type;
                          var status : smib_status_type);
    (* Given a smib index, this routine deletes the corresponding Active
       Association(s) and the security and key attributes that it refers to.
    *)
    primitive;

procedure PEERKEY_SEEK_AA (role: role_type;
                           peer_key_id : key_identifier_type;
                           var smib_id: smib_id_type;
                           var status : smib_status_type);
    (* Given the 'peer_key_id', this routine returns the index of the Active
      Association(s) with the matching peer key id.
    *)
    primitive;

procedure LOCALKEY_SEEK_AA (role: role_type;
                            local_key_id : key_identifier_type;
                            var smib_id: smib_id_type;
                            var status : smib_status_type);
    (* Given the 'local_key_id', this routine returns the index of the Active
      Association(s) with the matching local key id.
    *)
    primitive;

function permit_reuse(assoc: association_attributes_type;
                      ev: security_attributes_type;
                      key: key_attributes_type) : boolean;
    (* This routine determines if the indicated security association may
       be reused in a subsequent instance of communications.
    *)
    primitive;
```

```
procedure report_event( identifier : name_type;
                        severity : severity_type;
                        class : alarm_type;
                        reason : alarm_cause_type;
                        local_address : N_address_type;
                        peer_address : N_address_type);
    (* Used to notify systems management of any security errors encountered
       during key management processing.
    *)
    primitive;

initialize
    to IDLE
        begin
        end;

(* Transitions *)

trans

from IDLE to IDLE
    when K.ESTABreq
        provided available_SA_in_SMIB(role, local_address, peer_address,
                                      qos[protection_security])
            (* Request to establish a security association and one potentially
               is available in the SMIB.  Must first check access control
               facility for authorization, then retrieve the security
               attributes and key corresponding to the association, to be
               returned in the response.
            *)
            var
                retrieval_status, seek_status : smib_status_type;
            begin
            ADDRESS_SEEK_PA(role, local_address, peer_address,
                            qos[protection_security], smib_index, seek_status);
            if (seek_status = FOUND_PA_ATT) then
                if (AUTHassoc(role, smib_index, initiator) = AUTH_OK) then
                    begin
                    ADDRESS_SEEK_AA(role, local_address, peer_address,
                                    qos[protection_security], smib_index, seek_status);
```

```
        if (seek_status = FOUND_AA_ATT) then
            begin
            AA_info_retrieve(role, smib_index, assoc_attributes,
                             hold_security_attributes,
                             hold_key_attributes, retrieval_status);
            output K.ESTABresp(assoc_attributes.local_key_id,
                               assoc_attributes.peer_key_id,
                               hold_security_attributes, assoc_ok)
            end
        else
            begin
          report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                       ASSOCIATION_SEEK_FAILURE,
                       null_address, null_address);
            output K.ESTABresp(null_key_id, null_key_id,
                               null_security_attributes, assoc_failed);
            end
        end
    else
        begin
        report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                     AUTHORIZATION_FAILURE,
                     null_address, null_address);
        output K.ESTABresp(null_key_id, null_key_id,
                           null_security_attributes, assoc_failed);
        end
else
    begin
    report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                 ASSOCIATION_SEEK_FAILURE,
                 null_address, null_address);
    output K.ESTABresp(null_key_id, null_key_id,
                       null_security_attributes, assoc_failed);
    end;
end;
```

```
from IDLE to IDLE
    when K.VERIFYreq
            (* Verification of local key identifier conveyed by peer
               is requested.  Must find the active association
               corresponding to this key, check authorization,
               and return related information if verified.
            *)
            var
                seek_status, retrieval_status : smib_status_type;
            begin
            ADDRESS_SEEK_PA(role, local_address, peer_address,
                            qos[protection_security], smib_index, seek_status);
            if (seek_status = FOUND_PA_ATT) then
                if (AUTHassoc(role, smib_index, responder) = AUTH_OK) then
                    begin
                    ADDRESS_SEEK_AA(role, local_address, peer_address,
                                    qos[protection_security], smib_index, seek_status);
                    if (seek_status = FOUND_AA_ATT) then
                        begin
                        AA_info_retrieve(role, smib_index, assoc_attributes,
                                         hold_security_attributes,
                                         hold_key_attributes, retrieval_status);
                        if (local_key_id = assoc_attributes.local_key_id) then
                            output K.VERIFYresp(assoc_attributes.peer_key_id,
                                                hold_security_attributes, assoc_ok)
                        else
                            begin
                            report_event(my_identifier, MAJOR,
                                         OPERATIONAL_VIOLATION,
                                         AUTHORIZATION_FAILURE,
                                         null_address, null_address);
                            output K.VERIFYresp(null_key_id,
                                                null_security_attributes, assoc_failed);
                            end;
                        end
                    else
                        begin
                      report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                                   ASSOCIATION_SEEK_FAILURE,
                                   null_address, null_address);
                        output K.ESTABresp(null_key_id, null_key_id,
                                           null_security_attributes, assoc_failed);
                        end;
                    end
```

```
        else
            begin
            report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                        AUTHORIZATION_FAILURE,null_address,null_address);
            output K.VERIFYresp(null_key_id, null_security_attributes,
                              assoc_failed);
            end
    else
        begin
        report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                    ASSOCIATION_SEEK_FAILURE, null_address, null_address);
        output K.VERIFYresp(null_key_id, null_security_attributes,
                          assoc_failed);
        end;
    end;

from IDLE to IDLE
   when K.RELEASEreq
       (* Release of key requested.  Must find the corresponding active
          association and delete all related information, unless the
          association was externally established or other conditions,
          such as the key granularity, permits reuse.
       *)
       var
           retrieval_status, delete_status, seek_status : smib_status_type;
           release_status: crypto_status_type;
       begin
       LOCALKEY_SEEK_AA (role, local_key_id, smib_index, seek_status);
       if (seek_status = FOUND_AA_ATT) then
           begin
           AA_info_retrieve(role, smib_index, assoc_attributes,
                           hold_security_attributes,
                           hold_key_attributes, retrieval_status);
           if (assoc_attributes.type_of_assoc = EXT_ESTAB) or
              permit_reuse(assoc_attributes, hold_security_attributes,
                          hold_key_attributes) then
               output K.RELEASEresp
           else
               begin
               AA_info_delete(role, smib_index, delete_status);
               release_status := RELEASEtraffickey(cf_id, local_key_id);
               output K.RELEASEresp;
               end;
           end
```

```
        else
            begin
            report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                         ASSOCIATION_SEEK_FAILURE, null_address, null_address);
            output K.RELEASEresp;
            end;
        end;

from IDLE to WFPA
    when K.ESTABreq
        provided not available_SA_in_SMIB(role, local_address, peer_address,
                                        qos[protection_security])
        (* Request to establish a security association, but none is
           available in the SMIB.  Must first check access control
           facility for authorization.  Further processing not supported.
        *)
        var seek_status : smib_status_type;
        begin
        ADDRESS_SEEK_PA(role, local_address, peer_address,
                        qos[protection_security], smib_index, seek_status);
        if (seek_status = FOUND_PA_ATT) then
            if (AUTHassoc(role, smib_index, initiator) = AUTH_OK) then
                begin
                { on-line key establishment not supported in this version }
                report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                             UNSUPPORTED_FEATURE, null_address, null_address);
                output K.ESTABresp(null_key_id, null_key_id,
                                   null_security_attributes, assoc_failed);
                end
            else
                begin
                report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                             AUTHORIZATION_FAILURE,null_address,null_address);
                output K.ESTABresp(null_key_id, null_key_id,
                                   null_security_attributes, assoc_failed);
                end
        else
            begin
            report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                         ASSOCIATION_SEEK_FAILURE, null_address, null_address);
            output K.ESTABresp(null_key_id, null_key_id,
                               null_security_attributes, assoc_failed);
            end;
        end;
```

```
from IDLE to WFPA
    when K.REPLACEreq
            (* Requests to replace an existing security association are
               ignored.
            *)
        begin
            report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                        UNSUPPORTED_FEATURE, null_address, null_address);
            output K.REPLACEresp( null_key_id, null_key_id,
                                  null_security_attributes, assoc_failed);
         end;

from WFPA to WFAC
    when I2R.EXCHCREDresp
        provided (result = trans_ok)
            begin
            end;

from WFPA to IDLE
    when I2R.EXCHCREDresp
        provided (result <> trans_ok)
            begin
            end;

from WFAC to IDLE
    when I2R.SECATTRresp
            begin
            end;

end;   { end of KM_initiator body }

module KM_responder process(role: role_type; cf_id: cf_id_type);

    ip  R2I : KMGT_primitives (responder);
        (* This interaction point is used by a responder submodule of the key
           manager to communicate directly with the initiator side which lies
           within the peer key manager.
        *)

export  id : request_identifier_type;
end;
```

```
body kmr_body for KM_responder;

const
    my_identifier = kmresponder;

state IDLE, WFPA;

(* Functions and Procedures *)

function valid(credentials: credentials_type) : boolean;
    (* Check credentials for validity.
    *)
    primitive;

initialize
    to IDLE
    begin
    end;

(* Transitions *)

trans

from IDLE to IDLE
    when R2I.EXCHCREDreq
        provided valid(init_credentials)
            begin
            end;

from IDLE to IDLE
    when R2I.DELKEYreq
            begin
            end;

from IDLE to WFPA
    when R2I.EXCHCREDreq
        provided not valid(init_credentials)
            begin
            end;

from WFPA to IDLE
    when R2I.SECATTRreq
            begin
            end;
```

```
from WFPA to IDLE
    when R2I.ABORTreq
            begin
            end;

end;    { end of KM_responder body }

    ip  K: array [km_occurrence_type] of KMGR_primitives (user);
        (* This interaction point models the interface between the key
           manager and its submodules for multiplexing SP4_machine requests.
        *)

        I2R : array [km_occurrence_type] of KMGT_primitives (responder);
        (* This interaction point is used by an initiator submodule of the key
           manager to communicate with the key manager for multiplexing
           transactions onto a peer key manager.
        *)

        R2I : array [km_occurrence_type] of KMGT_primitives (initiator);
        (* This interaction point is used by a responder submodule of the key
           manager to communicate with the key manager for multiplexing onto
           a peer key manager.
        *)

modvar
    KMI : array [km_occurrence_type] of KM_initiator;
    KMR : array [km_occurrence_type] of KM_responder;

initialize
    to Active
    var
      i: integer;
    begin
    { clean up all outstanding temporary end-to-end associations }
    all i : km_occurrence_type do
        begin
        kmi_free[i] := true;
        kmr_free[i] := true;
        end;
    { get system credentials and load into crypto device }
    init_smib(role);
    load_cf(role, cf_id);
    GET_CERT(role, my_certificate, smib_status);
```

```
    if (smib_status = FOUND_CERT) then
        if (LOADcertificate(cf_id, my_certificate, null_key_id) <> LOAD_OK) then
            report_event(keymanager, CRITICAL, OPERATIONAL_VIOLATION,
                         CRYPTO_FACILITY_FAILURE,
                         Null_Address, Null_Address)
    else
        report_event(keymanager, CRITICAL, OPERATIONAL_VIOLATION,
                     CERTIFICATE_RETRIEVAL_FAILURE,
                     Null_Address, Null_Address);
    end;


(* Transitions *)

(* KEYservice multiplexing *)

trans

    any i : km_occurrence_type do
        when K[i].ESTABresp
            begin
            output KEYservice.ESTABresp(KMI[i].id, local_key_id, peer_key_id,
                                        security_attributes, status);
            disconnect K[i];
            disconnect I2R[i];
            release KMI[i];
            kmi_free[i] := true;
            end;

    any i : km_occurrence_type do
        when K[i].VERIFYresp
            begin
            output KEYservice.VERIFYresp(KMI[i].id, peer_key_id,
                                         security_attributes, status);
            disconnect K[i];
            disconnect I2R[i];
            release KMI[i];
            kmi_free[i] := true;
            end;
```

```
any i : km_occurrence_type do
    when K[i].RELEASEresp
        begin
        { release is not a confirmed service }
        disconnect K[i];
        disconnect I2R[i];
        release KMI[i];
        kmi_free[i] := true;
        end;

any i : km_occurrence_type do
    when K[i].REPLACEresp
        begin
        output KEYservice.REPLACEresp(KMI[i].id, local_key_id, peer_key_id,
                                    security_attributes, status);
        disconnect K[i];
        disconnect I2R[i];
        release KMI[i];
        kmi_free[i] := true;
        end;

trans

from Active to Active
    when KEYservice.ESTABreq
        begin
        forone i:km_occurrence_type suchthat kmi_free[i] do
            begin
            kmi_free[i] := false;
            init KMI[i] with kmi_body(role, cf_id);
            KMI[i].id := request_id;
            connect K[i] to KMI[i].K;
            connect I2R[i] to KMI[i].I2R;
            output K[i].ESTABreq(local_address, peer_address, qos);
            end
        otherwise
            report_event(my_identifier, MAJOR,
                         OPERATIONAL_VIOLATION,
                         MAX_KM_INSTANCES_EXCEEDED,
                         local_address, peer_address);
        end;
```

```
when KEYservice.VERIFYreq
    begin
    forone i:km_occurrence_type suchthat kmi_free[i] do
        begin
        kmi_free[i] := false;
        init KMI[i] with kmi_body(role, cf_id);
        KMI[i].id := request_id;
        connect K[i] to KMI[i].K;
        connect I2R[i] to KMI[i].I2R;
        output K[i].VERIFYreq(local_key_id, local_address,
                              peer_address, qos);
        end
    otherwise
        report_event(my_identifier, MAJOR,
                     OPERATIONAL_VIOLATION,
                     MAX_KM_INSTANCES_EXCEEDED,
                     Null_Address, Null_Address);
    end;

when KEYservice.REPLACEreq
    begin
    forone i:km_occurrence_type suchthat kmi_free[i] do
        begin
        kmi_free[i] := false;
        init KMI[i] with kmi_body(role, cf_id);
        KMI[i].id := request_id;
        connect K[i] to KMI[i].K;
        connect I2R[i] to KMI[i].I2R;
        output K[i].REPLACEreq(local_key_id);
        end
    otherwise
        report_event(my_identifier, MAJOR,
                     OPERATIONAL_VIOLATION,
                     MAX_KM_INSTANCES_EXCEEDED,
                     Null_Address, Null_Address);
    end;
```

```
        when KEYservice.RELEASEreq
            begin
            forone i:km_occurrence_type suchthat kmi_free[i] do
                begin
                kmi_free[i] := false;
                init KMI[i] with kmi_body(role, cf_id);
                KMI[i].id := request_id;
                connect K[i] to KMI[i].K;
                connect I2R[i] to KMI[i].I2R;
                output K[i].RELEASEreq(local_key_id);
                end
            otherwise
                report_event(my_identifier, MAJOR,
                             OPERATIONAL_VIOLATION,
                             MAX_KM_INSTANCES_EXCEEDED,
                             Null_Address, Null_Address);
            end;

(* TOpeer multiplexing *)

trans

from Active to Active
    any stream : K_STREAM_type do
        when TOpeer[stream].SECATTRresp
            begin
            forone i:km_occurrence_type suchthat
                (not kmi_free[i]) and (KMI[i].id = request_id) do
                begin
                output I2R[i].SECATTRresp(init_key_id,
                                          selected_options, result);
                end
            otherwise
                report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                             KM_INITIATOR_NOT_FOUND,
                             Null_Address, Null_Address);
            end;
```

```
from Active to Active
    any stream : K_STREAM_type do
        when TOpeer[stream].EXCHCREDresp
            begin
            forone i:km_occurrence_type suchthat
                (not kmi_free[i]) and (KMI[i].id = request_id) do
                begin
                output I2R[i].EXCHCREDresp(init_key_id, resp_key_id,
                                            resp_credentials, result);
                end
            otherwise
                report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                             KM_INITIATOR_NOT_FOUND,
                             Null_Address, Null_Address);
            end;

trans

from Active to Active
    any i : km_occurrence_type do
        when I2R[i].SECATTRreq
            var
                stream : K_STREAM_type;
            begin
            stream := determine_stream(KMI[i].id);
            output TOpeer[stream].SECATTRreq(KMI[i].id, resp_key_id,
                                    source_address, destination_address,
                                    proposed_options, resp_obsolete_key_id);
            end;

from Active to Active
    any i : km_occurrence_type do
        when I2R[i].DELKEYreq
            var
                stream : K_STREAM_type;
            begin
            stream := determine_stream(KMI[i].id);
            output TOpeer[stream].DELKEYreq(KMI[i].id, init_key_id);
            end;
```

```
from Active to Active
    any i : km_occurrence_type do
        when I2R[i].ABORTreq
            var
                stream : K_STREAM_type;
            begin
            stream := determine_stream(KMI[i].id);
            output TOpeer[stream].ABORTreq(KMI[i].id, init_key_id);
            end;

from Active to Active
    any i : km_occurrence_type do
        when I2R[i].EXCHCREDreq
            var
                stream : K_STREAM_type;
            begin
            stream := determine_stream(KMI[i].id);
            output TOpeer[stream].EXCHCREDreq(KMI[i].id, univ_id, init_key_id,
                                              init_credentials);
            end;

(* FROMpeer multiplexing *)

trans

from Active to Active
    any i : km_occurrence_type do
        when R2I[i].SECATTRresp
            var
                stream : K_STREAM_type;
            begin
            stream := determine_stream(KMR[i].id);
            output FROMpeer[stream].SECATTRresp(KMR[i].id, init_key_id,
                                                selected_options, result);
            disconnect R2I[i];
            release KMR[i];
            kmr_free[i] :=true;
            end;
```

```
from Active to Active
    any i : km_occurrence_type do
        when R2I[i].DELKEYresp
            begin
            { not a confirmed service multiplexed to peer }
            disconnect R2I[i];
            release KMR[i];
            kmr_free[i] :=true;
            end;

from Active to Active
    any i : km_occurrence_type do
        when R2I[i].ABORTresp
            begin
            { not a confirmed service multiplexed to peer }
            disconnect R2I[i];
            release KMR[i];
            kmr_free[i] :=true;
            end;

from Active to Active
    any i : km_occurrence_type do
        when R2I[i].EXCHCREDresp
            var
                stream : K_STREAM_type;
            begin
            stream := determine_stream(KMR[i].id);
            output FROMpeer[stream].EXCHCREDresp(KMR[i].id, init_key_id,
                                        resp_key_id, resp_credentials, result);
            { retain module since further exchanges are expected }
            end;
```

trans

```
from Active to Active
    any stream : K_STREAM_type do
        when FROMpeer[stream].EXCHCREDreq
            begin
            forone i:km_occurrence_type suchthat kmr_free[i] do
                begin
                kmr_free[i] := false;
                init KMR[i] with kmr_body(role, cf_id);
                KMR[i].id := request_id;
                connect R2I[i] to KMR[i].R2I;
                output R2I[i].EXCHCREDreq(univ_id, init_key_id, init_credentials);
                end
            otherwise
                report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                             MAX_KM_INSTANCES_EXCEEDED,
                             Null_Address, Null_Address);
            end;

    any stream : K_STREAM_type do
        when FROMpeer[stream].DELKEYreq
            begin
            forone i:km_occurrence_type suchthat kmr_free[i] do
                begin
                kmr_free[i] := false;
                init KMR[i] with kmr_body(role, cf_id);
                KMR[i].id := request_id;
                connect R2I[i] to KMR[i].R2I;
                output R2I[i].DELKEYreq(init_key_id);
                end
            otherwise
                report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                             MAX_KM_INSTANCES_EXCEEDED,
                             Null_Address, Null_Address);
            end;

    any stream : K_STREAM_type do
```

```
        when FROMpeer[stream].SECATTRreq
            begin
            forone i:km_occurrence_type suchthat
                (not kmr_free[i]) and (KMR[i].id = request_id) do
                begin
                output R2I[i].SECATTRreq(resp_key_id, source_address,
                                         destination_address, proposed_options,
                                         resp_obsolete_key_id);
                end
            otherwise
                report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                             KM_RESPONDER_NOT_FOUND,
                             Null_Address, Null_Address);
          end;

    any stream : K_STREAM_type do
        when FROMpeer[stream].ABORTreq
            begin
            forone i:km_occurrence_type suchthat
                (not kmr_free[i]) and (KMR[i].id = request_id) do
                begin
                output R2I[i].ABORTreq(init_key_id);
                end
            otherwise
                report_event(my_identifier, MAJOR, OPERATIONAL_VIOLATION,
                             KM_RESPONDER_NOT_FOUND,
                             Null_Address, Null_Address);
            end;

end;        { end of kmgr module body }
```

APPENDIX D
ATTRIBUTE DEFINITIONS FOR SMIB OBJECTS

D.1. Authentication Objects

(1) System

(a) system identifier - the name of the system.
(b) policy identifier - the name of the security policy in effect for the system.
(c) defining authority - the identity of the authority who defined the security policy.

(2) Certificate

(a) system certificate - the signed identification information for the system as distributed by a key management center on behalf of the certification authority.
(b) universal identifier - identifier of the security domain associated with this certificate.

D.2. Authorization Objects

(1) Permitted Association

(a) permitted association identifier - the name of this permitted association instance.
(b) local address designator - the local address of an association pair.
(c) peer address designator - the peer address of an association pair.
(d) initiated by - the identity (i.e., local, peer, or either) of the system that may initiate a protected connection.
(e) day/time restrictions - the day/time periods in which an association may be established.

(2) Constraint

(a) permitted association identifier - the name of the permitted association instance to which these restrictions apply.
(b) negotiable services - indication of the security services that may be negotiated:
    i) confidentiality (optional, mandatory)
        a) permitted algorithm identifiers

- ii) integrity (optional, mandatory)
  - a) permitted algorithm identifiers
  - b) permitted ICV sizes
- iii) key granularity (by security level, end-to-end)
- iv) security labeling (optional, mandatory)
  - a) permitted defining authority
  - b) permitted security level range (min, max)
  - c) permitted security category range (min, max)

D.3. Access Control Objects

(1) Active Association

- (a) active association identifier - the name of this active association instance.
- (b) local address designator - the local address of an association pair.
- (c) peer address designator - the peer address of an association pair.
- (d) initiated by - determines which end-system sets the direction indicator as the initiator (local or peer).
- (e) status - the status of the association (partially complete, usable, etc.).
- (f) local credentials - the credentials used to establish this association.
- (g) peer credentials - the peer entities' credentials used to establish this association.
- (h) local key identifier - the identifier of the traffic key used for this association.
- (i) peer key identifier - the peer entities identifier for the traffic key used for this association.

(2) Restraint

- (a) active association identifier - the name of the active association instance to which this enforcement vector applies.
- (b) negotiated services - the services negotiated for this association:
  - i) confidentiality (yes, no)
    - a) algorithm identifier
  - ii) integrity (yes, no)
    - a) algorithm identifier
    - b) ICV size
  - iii) key granularity (by security level, end-to-end)
  - iv) security labeling (yes, no)
    - a) defining authority
    - b) security level
    - c) security category

(3) Key

(a) local identifier - the identifier associated with the traffic key material (tek-id).
(b) status - the status of the traffic key (e.g., compromised, expired).
(c) key material - the values of the generated traffic key and any associated initialization variables.
(d) cryptographic period - the start and duration values of the period in which the traffic key is valid.
(e) application - the context for which this key was generated:
    i) confidentiality (yes, no)
        a) algorithm identifier
    ii) integrity (yes, no)
        a) algorithm identifier
        b) ICV size
    iii) key granularity
    iv) security label

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4792

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
MARCH 1992

4. TITLE AND SUBTITLE

A Formal Description of the SDNS Security Protocol at Layer 4 (SP4)

5. AUTHOR(S)

Wayne A. Jansen

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)
National Institute of Standards and Technology
Computer Systems Laboratory
Bldg. 225/Rm. A216
Gaithersburg, MD 20899

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

The Secure Data Network System (SDNS) project, initiated by the National Security Agency in 1986, produced a computer network security architecture within the framework of the International Organization for Standardization (ISO) reference model for Open Systems Interconnection (OSI). This report contains a formal description of the SDNS security protocol at layer four (SP4), one component of the overall security architecture. Estelle is the OSI formal description technique (FDT) used for the SP4 specification. Estelle is based on an extended state transition model with language elements from the Pascal language. An Estelle specification describes a hierarchically structured system of modules. The design of the formal description is explained through a top level and subsequent level of module decomposition. A description of the underlying security management information base is also included.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

Computer Network Security; Formal Description Technique; Open Systems Interconnection; Secure Data Network System; Security Protocol

13. AVAILABILITY

- [X] UNLIMITED
- [ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).
- [ ] ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.
- [X] ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
108

15. PRICE
A04

ELECTRONIC FORM