Department of Computer Science

Kang Chung
Mathilda Gustafsson

Prototyping and evaluation of TCAPsec

Degree Project of 20 credit points
Master's in Computer Science and Engineering

Date:	2006-02-01
Supervisor:	Katarina Asplund
Examiner:	Donald F. Ross
Serial Number:	D2007:04

Karlstad University 651 88 Karlstad
Telephone 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

Prototyping and evaluation of TCAPsec

Kang Chung

Mathilda Gustafsson

This report is submitted in partial fulfilment of the requirements for the Master's degree in Computer Science. All material in this report which is not our own work has been identified and no material is included for which a degree has previously been conferred.

Kang Chung

Approved, February 1, 2006.

Advisor: Katarina Asplund

Examiner: Donald F. Ross

Abstract

Today, the most frequently used signaling system for telecommunication is called Signaling System No. 7 (SS7). The growing usage of mobile telephones and mobile data communication, and the development of new services mean that the risk of intrusion and exploitation of the SS7 signaling networks increases. The increasing problem with unauthorized access to sensitive information and the operators' growing demand for security is the origin of our work. This thesis presents a prototype design and implementation of a Security Gateway (SEG), which is a fundamental part of the TCAP user security (TCAPsec) concept. TCAPsec is a security concept for introducing security mechanisms to the signaling system. The prototype includes three different protection modes that provide security services, ranging from almost no protection to full protection with the use of encryption algorithms. The thesis also contains an evaluation study of the delay penalties caused by the use of these security services. With regards to the restrictions on the prototype, the conclusion drawn from the evaluation results was that the protection mechanisms in the different protection modes did not inflict any significant time penalties. Instead, the results of the study indicate that the routing process of messages in the network is a more significant delaying part in the communication between different nodes. This result implies that the routing process takes longer time than the security services. The thesis also presents a number of discovered features that will require further investigation and development before the TCAPsec concept can be realized.

Acknowledgements

The project was undertaken at TietoEnator R&D Services AB in Karlstad. The project work was performed jointly with Mathilda Gustafsson, who will also be presenting a separate report to Mid Sweden University in Sundsvall as a Master's degree project.

We would like to give thanks to everyone who has helped us during our Master's thesis. First and foremost we would like to thank Robert Locke and Daniel Rååd, our supervisors at TietoEnator, for sharing their knowledge and guiding us in the right direction when problems occurred. We would also like to give particular thanks to Rafael Espino, Robert Locke, Karl-Johan Grinnemo and Ulf Melin, from who we had the benefit to borrow books.

We would also like to express our gratitude to Lars Ahlin and Peter Torpman for their programming help and to Malin Abrahamsson and Rafael Espino for their encouragement and support. Furthermore, we would like to give thanks to Karl-Johan Grinnemo for his helpfulness and deep commitment during our thesis work at TietoEnator.

Special thanks go to our supervisor Katarina Asplund at the Department of Computer Science, Karlstad University, for devoting so much time for helping us with our thesis.

Last, but not least, we would like to thank Anna Brodd for giving us the opportunity of performing our Master's thesis at TietoEnator.

Table of Contents

List of Figures

List of Tables

1 Introduction

The global telecommunication network could arguably be the largest and most complex technical system that has ever been created. This system is important for people's local to global communication and for countries' development worldwide. In recent years, the development of telecommunication has accelerated.

A telecommunication network is, opposite to a computer network, made up of two different networks. The first part of the network is responsible for carrying voice, video and data traffic. The second part handles control information and is known as the signaling network. The main purpose of the signaling network is to transfer essential information for managing connections between different nodes in a network.

Today, the telecommunication signaling system called *Signaling System No. 7* (SS7) is probably the most used signaling system. It is a standard that is used in all modern telecommunication networks. Signaling in ordinary telephony is relatively uncomplicated in opposite to mobile telephony, where the system has to retain location information on the mobile telephones. The growing usage of mobile telephones and mobile data communication, and the development of new services mean that the demand for capacity, flexibility, rate and security in signaling networks increase. Because of the increased communication in networks, network security is a hot topic these days. More sensitive information is sent in the networks (such as bank transaction information), which in turn could attract fraudulent actors that exploits the system. A number of telephone operators have been experiencing an increase in *Short Message Service* (SMS) frauds in 2004 (described in Section 2.5). The growing problem with unauthorized access to sensitive information is the origin of our work. To counter the increasing number of frauds, security mechanisms were introduced to the *Transaction Capabilities Application Part* (TCAP) in SS7 to form a concept called *TCAP user security* (TCAPsec).

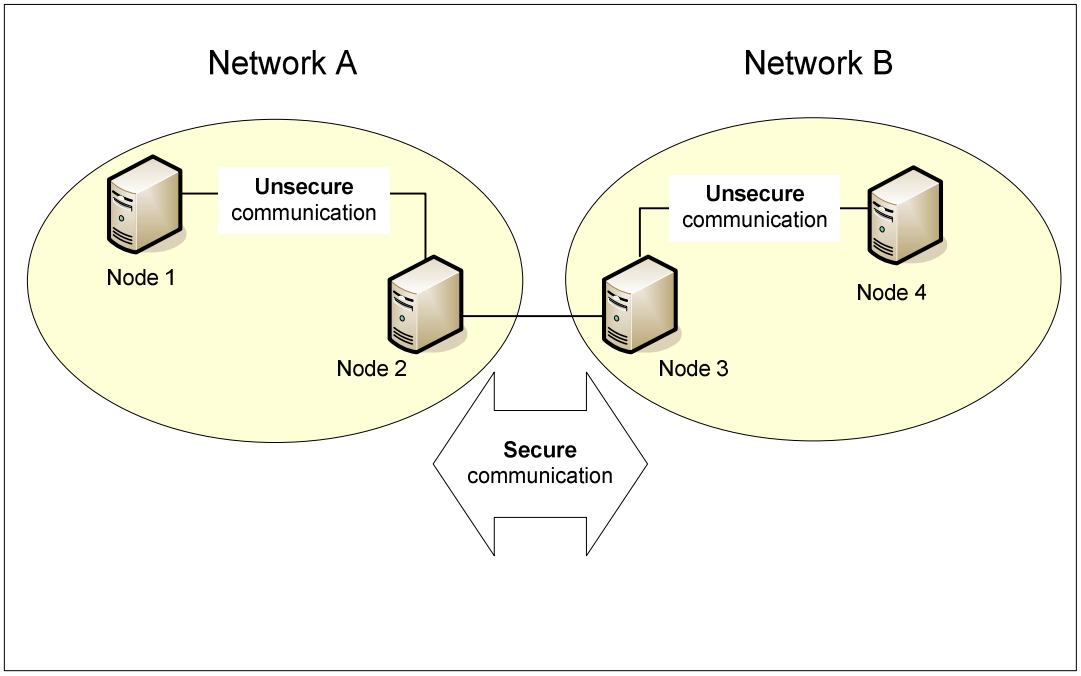


Figure 1.1: Secure inter-network communication.

The TCAPsec concept includes different protection modes to provide different levels of protection when protecting the inter-network communication, see Figure 1.1. These protection modes provide security services ranging from almost no protection to full integrity and encryption protection with the use of standardized encryption algorithms. By applying services such as integrity calculations and encryption on the routed messages, SMS-spamming can be prevented and discouraged. The thesis also contains an evaluation study of the delay penalties caused by these protection modes when introducing TCAPsec.

1.1 Purpose

The purpose of this thesis will be to perform an assignment given by TietoEnator. The assignment consisted of a problem statement with two major parts.

1. Investigate how a *Security Gateway* (SEG, part of the TCAPsec concept) could be realized with TietoEnator's SS7 signaling stack.

2. Evaluate the delay penalties caused by the encryption/decryption mechanisms when introducing TCAPsec.

To solve the problems stated, the thesis work was executed in two steps.

1. A prototype of a SEG was built. Since the primary incentive with the prototype was to evaluate performance penalties, the only part of the prototype that had to be realistically implemented is the encryption/decryption function. The rest of the prototype could be built around stubs.

2. An evaluation study was conducted. The study considers the delay penalties inherent with various encryption and integrity algorithms.

Including the above described steps, the thesis work consists of three major parts. At first we describe TCAP user security (TCAPsec) and the purpose of introducing TCAPsec in telecommunication networks. The second part is to examine the feasibility of implementing TCAPsec by designing a prototype of a SEG. The third part of our work is to perform an evaluation study of the penalties due to encryption/decryption of introducing TCAPsec.

1.2 Outline

This thesis is organized in the following way. Chapter 2 presents background information for telecommunication and Signaling System no. 7 (SS7). Also, information regarding the Open System Interconnection (OSI) model is included for comparison, together with a brief presentation of network security and the motivation for the existence of the TCAPsec concept. Chapter 3 contains the description of the TCAP user security (TCAPsec) concept. Chapter 4 contains the requirements and the design of the TCAPsec prototype. In Chapter 5 the performance evaluation is presented with a discussion of the results. Chapter 6 contains a presentation of the encountered problems and a discussion of the experiences we have gained during this thesis work. Finally, in Chapter 7, we present our conclusions, and a number of investigation and evaluation recommendations for future work.

This thesis contains a large number of abbreviations that can be found in Appendix A.

1.3 Contributions

This Master’s thesis is performed by Kang Chung, for Karlstad University, and Mathilda Gustafsson, for Mid Sweden University. The planning and designing of the TCAPsec proto-

type was a joint effort. The work was divided so that the different parts of the work could be done in parallel. The thesis consists of a number of chapters, where the main responsibility for each chapter is solely placed on one author or divided between the authors. Help has been offered by the co-author, but the main responsibility for a single chapter is assigned to one of the authors. The responsibility was divided as follows. Gustafsson had the main responsibility for Abstract, Acknowledgement, Abbreviations, References and the contents of the Appendix A to C. Furthermore Gustafsson was responsible for Chapter 1, and 5. Chung had the main responsibility for Chapter 4, 6, 7, and Appendix D to E. The responsibility for Chapters 2 and 3 were divided between the authors.

TietoEnator provided us with the hardware and software required to perform the thesis work. Two computers were configured as two stacks. This configuration was performed by Lars Ahlin at TietoEnator. A program skeleton containing the callback functions required by the TCAP module was provided by our supervisor at TietoEnator, Daniel Rååd. The implementation was also divided up between the two authors and involved approximately four weeks work for each. Gustafsson was responsible for implementing the routing between the end nodes via the two SEGs. Chung was then responsible for implementing the security mechanisms into the prototype SEGs. The implementation results are the property of TietoEnator.

1.4 Post-opposition changes

After the opposition on Karlstad University on the 1st of February 2006, the following changes to the thesis were requested by the examiner Donald Ross and performed by Chung:

- A summary were added to the end of chapters 2, 3, and 4 to facilitate the reading of the thesis.

- Subchapter 1.3 were added for separating and documenting the separate responsibilities of the authors.

- Subchapter 1.4 were added for documenting the changes made in the thesis by Chung after the opposition in Karlstad University.

2 Background

This chapter gives a background to the project. Section 2.1 describes the history of telecommunication and signaling. In Section 2.2 the Signaling System no. 7 is explained. Section 2.3 contains information about the Open System Interconnection (OSI) model. This model is further compared with both the TCP/IP model and the SS7 model. The SS7 model is composed of the layers MTP, SCCP, TCAP and MAP, which are described later in Section 2.3. Section 2.4 presents a brief overview on network security and contains both information about fraudulent actors and the protection against them. Section 2.5 completes this chapter and describes the motivation behind the TCAPsec concept and this thesis. In this section, the scenario of SMS spamming is explained in detail.

2.1 Telecommunication

Antonio Meucci [1] developed the first modern telephone in the late 1840s. This construction was used for connecting his bedroom and his office. Unfortunately, Meucci was destitute and could not pay for the patent application. At the same time, a boy named Alexander Graham Bell [2] was born in England. In the early 1870s, Bell made a working telephone with both a sender and a receiver in the same unit. Since Meucci could not afford the patent application, Bell was in the year 1876 free to apply for a patent on his invention. Ever since this day in 1876 the *Public Switched Telephone Network* (PSTN) has been under constant development.

In early telephony, numbers couldn't be used when contacting a specific person [3, 4, 5]. Instead, a cable had to be directly connected between the sender's and the receiver's equipments. This means that everyone has to have one cable connected per person he/she wishes to speak with. This solution was naturally both extensive and costly and therefore new solutions were in demand. One solution was the switch. When the switch was introduced, only one cable was needed to connect the equipment to the switch. The switch then put the conversation through using other cables connected to it.

At the beginning, the switch was operated by telephone operators that asked the callers who they wanted to talk with [3, 4, 5]. The telephone operators then connected the cables attached to the two persons' equipment and they could start a conversation. This was an early sort of signaling. A number of years later, automatic switches took over the connecting part. This automatization had an effect on the signaling information[1], which was moved to a separate network inside the PSTN-network. This split meant that the conversation and the signaling information were transmitted in separate channels.

2.1.1 Signaling

Signaling [3, 4] in a technical context is about controlling processes. The main purpose with signaling in modern telecommunication networks is to transfer control information between nodes.

In former times, signaling between the subscriber and the local station was manual. When the subscriber turned the handle on its telephone unit, an alternating voltage was generated and a signal was triggered at the telephone operator. The telephone operator and the subscriber then orally exchanged the number to the receiver. This information was then exchanged between telephone operators in different stations.

When automation of the telecommunication network was introduced, the subscriber no longer needed to perform this oral exchange of information with a telephone operator before a call could be established. The telephone was instead equipped with a hook and a rotary dial. The communication between the subscriber and the local station has since that day been performed by different tones, for example dial tone, ring tone and occupied tone.

Telephony has changed over time. Earlier, the communication was made up of pulses and tones. But in today's digital networks, packets of binary data are used for carrying signals. The exchange of control information is a form of data communication. In signaling, there are both similarities and differences in related services (between data communication and telecommunication), but the main purpose is to transfer essential information for handling the telecommunication connection. Signaling in ordinary telephony is relatively uncomplicated in oppo-

[1] That is, the information that is needed for connecting and routing of a telephone call between the correct sender and receiver.

site to mobile telephony, where the system, for example, has to maintain location information of the mobile telephones. This information is stored in a special database that is called *Home Location Register* (HLR) [3], see MAP in Section 2.3.2 for more details.

The increased usage of mobile telephones and mobile data communication, and the development of new services mean that the demand for capacity, flexibility, rate and security in signaling systems increase. One well-developed signaling system is the *International Telecommunication Union's* (ITU's) [4] SS7, which is described in Section 2.2 below.

2.2 Signaling System no. 7 (SS7)

International Telecommunication Union (ITU) [5, 6] created a standard in 1980 for digital signaling communication. This standard is called Signaling System no.7 (SS7) [3, 4, 5] and is a global standard for signaling in telecommunication networks. The reason behind the development of SS7 was to make the network more efficient and maximize the utilization of the resources. The development of SS7 led to a considerably improved performance in the PSTN.

SS7 is a reliable packet-switched data network that has an important part in both wired and wireless networks. SS7 is also a separate network for transmission of control information, working in an existing voice network.

The SS7 network consists of three main components as shown in Figure 2.1. Telephone switches connected by SS7 links are called *Service Switching Points* (SSP). SSP manages calls and sends SS7 messages for transferring conversation related information to other SSPs. A conversation can be routed only if the *Service Control Point* (SCP) supplies the SSP with routing information.

A *Signal Transfer Point* (STP) is a switch that forwards messages between network switches and databases. The messages are routed to the correct signaling link using information found in SS7 data messages.

SCP is a database containing central network databases designed to make reliable information available. SCP receives a request from a SSP and returns the information that was asked for. We can take the 020-number for example. When a 020-number is called, the SCP checks the

number in a table and gives SSP the actual number for routing the conversation. This number can vary depending on, for example, day and time.

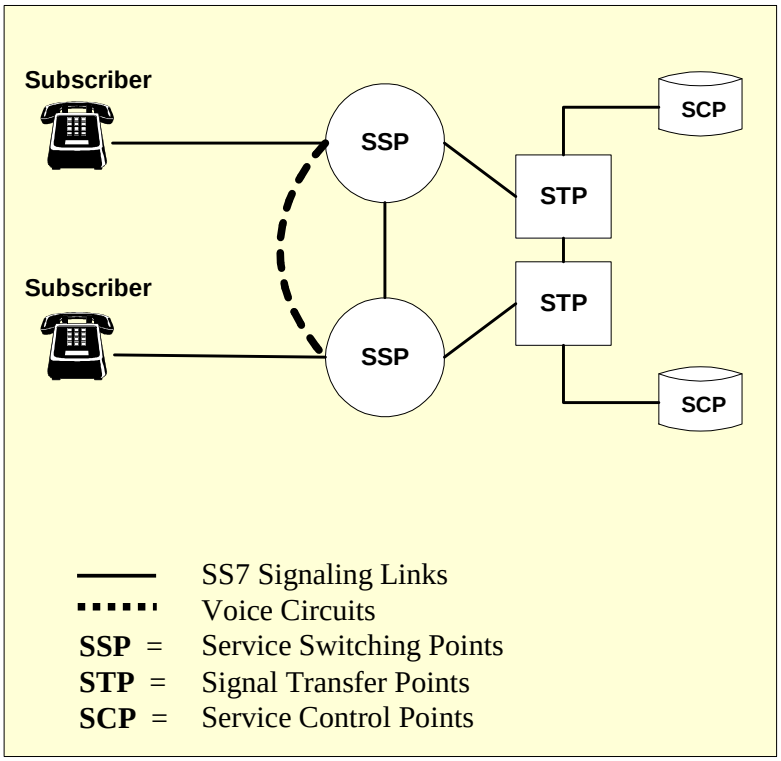


Figure 2.1: Main components in a SS7 signaling network [6].

In the SS7 network, *Signaling Points* (SPs) [3, 4, 6, 7] are communicating by using signaling links. The SPs are identified using a *Signaling Point Code* (SPC) address, unique for each network. Every message that is sent in the network has an originating address and a destination address. These addresses are called *Originating Point Code* (OPC) and *Destination Point Code* (DPC). By using these addresses the messages will be sent to the correct destination.

Since SS7 has a large transmission capacity, all nodes do not need to be connected with signaling links. Figure 2.2 below shows that station A and station B can communicate without having a direct signaling link between the stations. Each of them has a direct link to station C and the signaling between the two stations can therefore be routed through station C.

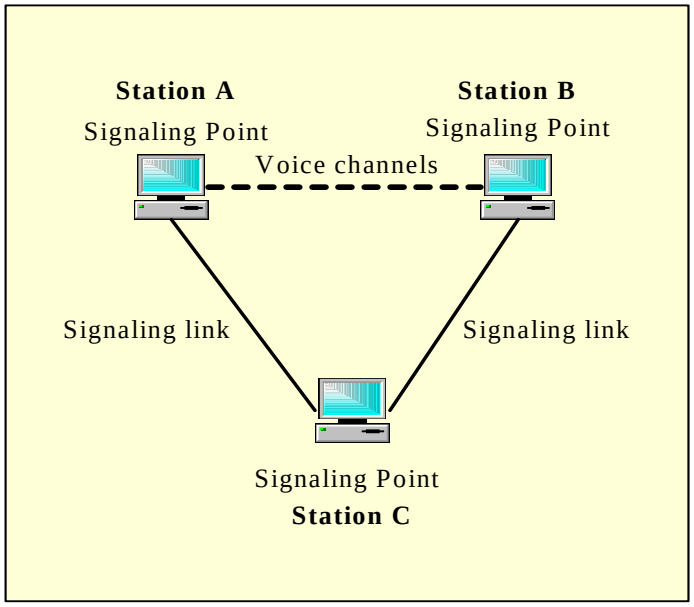


Figure 2.2: Communication between station A and B using signaling links [3].

The SS7 software is designed as a protocol stack. To facilitate the understanding of the stack, the SS7 stack and the *Transmission Control Protocol/Internet Protocol* (TCP/IP) stack is briefly compared with the *Open System Interconnection* (OSI) stack, which is a commonly approved protocol stack and is described in Section 2.3. TCP/IP is compared in Section 2.3.1 and SS7 in Section 2.3.2.

2.3 Open System Interconnection (OSI)

The need for communication between different computer systems led to the development of an international standard. This development work began in 1977 by the *International Standard Organization* (ISO) [3, 4, 8, 9]. The goal was to create a standardized interface for interconnecting computer communication systems around the world. The standard they presented was called Open System Interconnection (OSI) [3, 4, 8, 9].

One purpose with the development of OSI was to facilitate the work and the communication between different companies. By using a standard, compatibility issues between different implementations could be eliminated. The OSI model could be used by companies that use the same interface, but have individual implementations.

A protocol stack consists of several layers of software where each layer provides the upper layers with services. Each layer has its own service responsibilities. The protocol stack of the OSI model consists of seven layers, as shown in Figure 2.3 below.

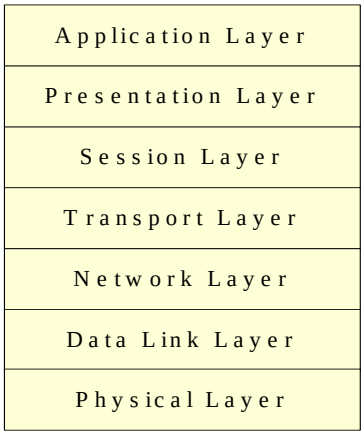


Figure 2.3: The OSI protocol stack.

The top layer is called the application layer, see Figure 2.3 above, and it provides the users' applications with network services and controls the communication between distributed applications.

The main task for the presentation layer, see Figure 2.3, is to translate application data to a common syntax. Translation is necessary, otherwise the applications cannot communicate.

The session layer, see Figure 2.3, establishes communication between presentation layers in different systems. It maintains, synchronizes and tears down the actual dialogue. The session layer renders the possibility for the presentation layer to calculate a checkpoint in the data-stream for resending if the connection is broken.

The transport layer, see Figure 2.3 above, supplies process-to-process communication. The transport layer guarantees that the carrier fulfils the current application's requirements. Functions in the transport layer are for instance flow control, error-detection and correction. The transport layer also optimizes the data communication by using multiplexing, or dividing data into smaller packets before it reaches the network.

The network layer's (see Figure 2.3) main function is to transport data packets between hosts in the network. The network layer decides the path taken by packets on their way from sender to receiver. This layer is accordingly responsible for the addressing of hosts.

The data link layer (see Figure 2.3) is responsible for moving a data-packet between adjacent nodes through an individual link in the network. To clarify this, the network layer is responsible of moving packets from the transport layer at the sending node to the transport layer at the receiving node.

Layer one is called the physical layer (see Figure 2.3). It contains functions for converting data to signals to make it compatible with the transmission media.

The OSI model is used as a reference model when creating other protocol stacks. Today, there are several protocol stacks that are implemented in a similar fashion as the OSI model. Two of these stack models are the TCP/IP- and the SS7-model, which are described in Section 2.3.1 and Section 2.3.2.

2.3.1 The TCP/IP model

The Transmission Control Protocol/Internet Protocol (TCP/IP) [8, 10] model is a protocol stack, implemented in a similar approach as the OSI model. See Figure 2.4 below.

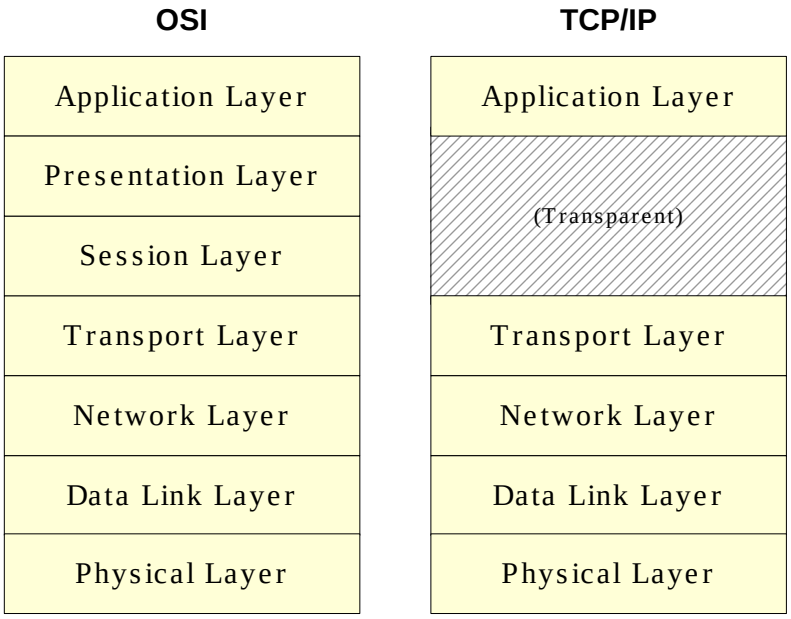


Figure 2.4: A side-by-side layer comparison of the OSI stack and the TCP/IP stack.

Neither the presentation layer nor the session layer exists in the TCP/IP model. The TCP/IP model provides circuitless packet-switched communication. The network layer supplies a best effort service.

Protocols that work in the TCP/IP stack are, for instance, *Hypertext Transfer Protocol* (HTTP) [11], *Simple Mail Transfer Protocol* (SMTP) [12] and *File Transfer Protocol* (FTP) [13] in the application layer. Two protocols that work in the transport layer are *Transmission Control Protocol* (TCP)[14] and *User Datagram Protocol* (UDP) [15]. TCP is used when a reliable transport of messages is required or favourable, for example when sending e-mail. In the situation when speed is more desirable than reliability, UDP is used; an example is when streaming multimedia. The protocol that works in the network layer is the *Internet Protocol* (IP), which supplies a best effort service. Protocols used in the data link layer vary and depend on the type of network used.

2.3.2 The SS7 model

When SS7 [5] was introduced, telecommunication became packet-switched and circuitless. The SS7 model was created nearly at the same time as the OSI model, but SS7 was developed for use in telecommunication and therefore the terminology used is very different, see Figure 2.5 below.

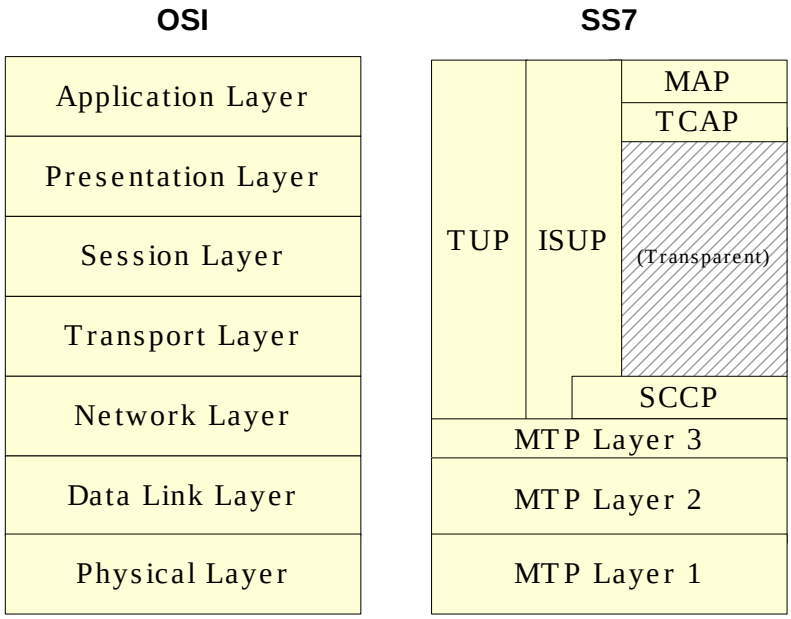


Figure 2.5: A side-by-side layer comparison of the OSI stack and the SS7 protocol stack.

Each layer in the SS7 stack has a specific role. The OSI-model layers 1-3 represent functions for transferring information between different nodes. These functions essentially constitute the communication network. *Message Transfer Part* (MTP) [5, 10] and *Signaling Connection Control Part* (SCCP) are, as is shown in Figure 2.5 above, equivalent to layer 1-3 in the OSI-model. Layers 4-7 in the OSI-model define functions related to end-to-end communication. The SS7 protocol stack does not have any protocols that can be mapped directly into the OSI layers 4-6. However, some protocols, such as the *Telephony User Part* (TUP) and *ISDN User Part* (ISUP), covers their functionality to some extent. The TUP, ISUP, Transaction Capabilities Application Part (TCAP), and *Mobile Application Part* (MAP) supply some of the services in the OSI-model's layer 7.

ISUP [5, 10] and TUP are circuit-related signaling protocols. They are responsible for the signaling when setting up, maintaining and tearing down telephone calls. TUP only support old telephone service and has in most cases been replaced by ISUP, which supports both telephone calls and *Integrated Services Digital Network* (ISDN) -calls. We will not describe these two protocols further, as they are not within the scope of our work.

The following subsections provide an outline of the SS7 protocols found in the SS7 model, see Figure 2.5.

Message Transfer Part (MTP)

The Message Transfer Part (MTP) [4, 5, 7] is a transport protocol that is used in SS7-networks. It is responsible for the transmission of signal units and it provides an error-free communication between two SPs.

As shown in Figure 2.5, MTP consists of three parts, which cover the first three layers of the OSI stack (Physical-, Data Link- and Network layer). MTP layer one (MTP1) represents the OSI's physical layer and it has full responsibility for the connection between the SPs and the transmission network. This layer also decides which physical link that is going to be used during the transmission.

MTP layer two (MTP2) works under MTP layer three (MTP3) and follows directions from that layer. MTP2 supplies a reliable transmission of signaling information between the SPs in the network. To achieve this, MTP2 uses error-detection and error-correction, but also flow control. An advantage in MTP2 is the possibility of dividing outgoing messages into smaller packets.

MTP layer three (MTP3) is responsible for message routing and the related network selection. It distributes incoming messages to upper layers, for example TUP, ISUP and SCCP, and it routes outgoing messages toward their destination. MTP3 uses routing control and error handling when the messages are directed to the correct destination. Each message has both an OPC and a DPC. Based on this information, MTP3 can deliver the messages to the correct destination.

MTP3 is also responsible for selecting the best link for transmitting messages. This is done by distribution of messages out on different available links. In this approach, congestion is avoided in a particular link, and if the link will fail, MTP3 is able to handle errors.

Signaling Connection Control Part (SCCP)

The Signaling Connection Control Part (SCCP*)* [4, 7, 16] is located above the MTP in the SS7 signaling stack and is, together with the MTP, called the *Network Service Part* (NSP). As the signaling network evolved, new requirements arose and new protocols were developed to meet these requirements. SCCP is a prime example of such a protocol. The SCCP implements two kinds of transfer capabilities: circuit related signaling and non-circuit related signaling. The circuit related signaling is not that commonly used, as the MTP was designed to do this. Instead, the main reason for this part's existence is the non-circuit related signaling. This way of signaling implies that, unlike a common phone call that requires a path or a circuit to be exclusively allocated, communication is achieved between the end nodes without creating an exclusive circuit. An example of usage for non-circuit related signaling is data communication between databases. Such communication implies short data bursts and quick queries, for example location updates for mobile stations or cellular phones.

SCCP also introduces two modes of communication for signaling: *connectionless* (CL) mode and *connection-oriented* (CO) mode. These modes are similar to UDP and TCP (described in Section 2.3.1) in an IP-based network. In CL mode, the packets contain all the information to route the packets without setting up a logical connection before sending the packet (similar to the UDP traffic in TCP/IP-based networks). This mode is favourable for short and bursty traffic such as short database queries. Contrary to the CL mode, the CO mode requires an initial set-up phase for creating a logical connection between the end nodes (similar to TCP). During the set-up phase, a local reference number is stored in the nodes along the set route. Using this reference number for all the messages belonging to a specific connection makes the routing faster than in the CL mode. The speed advantage makes the CO signaling mode more suitable for larger packets and for larger number of packets, such as maintenance communication for downloading test results from nodes.

As a node is capable of having several different applications and users (for example databases), the SCCP was designed to handle the distribution of incoming messages to the differ-

ent users in the upper layer. Here, SCCP introduces the ability to distinguish between several upper layer applications, using a *Subsystem Number* (SSN) to identify each application or user in the terminating nodes/end nodes.

One of the most important functions of SCCP is the *Global Title Translation* (GTT) using *Global Titles* (GT). As MTP uses Originating Point Code (OPC) and Destination Point Code (DPC, see Section 2.2) within a single *Public Land Mobile Network* (PLMN, described in MAP below), SCCP complements this with GTs to communicate with other PLMNs. A *Point Code* (PC) addressing a single node is unique within a PLMN. But the same PC address can reoccur in other PLMNs, causing confusion when communicating across more than one PLMN. The solution to the inter-PLMN communication is the GT that is unique for all nodes globally by identifying a node with both a network id and the PC. A simplified description of the GTT is that it translates a GT to a PC and SSN for routing within the operator network.

Transaction Capabilities Application Part (TCAP)

The Transaction Capabilities Application Part (TCAP) [4] is located in the application layer according to the OSI model. As new services are developed, so are the needs for faster and more efficient data transfers of larger amounts without going through the process of allocating a circuit. The main purpose of TCAP is to provide the upper applications with generalized services to send information over the network using dialogues and non-circuit related signaling. Before TCAP was developed, each application implemented its own dialogue handling separately. Therefore, a standardized dialogue handling function was created to minimize the need for creating new services and protocols to handle dialogues. TCAP provides the services to transfer information between nodes, independent of applications. TCAP is defined as an end-to-end protocol, which implies that the protocol will not process the message other than in the sending and receiving end nodes. Also, the non-circuit related signaling that TCAP uses implies that the protocol requires the support and services of the NSP (SCCP and MTP combined) to function properly.

The functionality in TCAP requires the introduction of a number of terms, such as the following:

- **TC user:** denotes the application that uses TCAP’s services.

- **Dialogue:** an association created between two TC users for transferring messages.
- **Dialogue ID:** an identifier to separate a dialogue from other dialogues. The dialogue id is a value between zero and 65536. The value is set to low when sending down the stack and high when receiving from the stack.
- **Transaction:** an association created between two TCAP protocols in different nodes.
- **Transaction ID:** an identifier to associate a message to the above described transaction.
- **Operation:** an action requested by the local TC user for the remote end to perform.
- **Component:** a data unit that is sent between two TC users, normally containing an operation request or result.
- **TC primitive:** a message sent internally between the TC user and TCAP. The primitive is either a *request*, sent from the TC user to TCAP, or an *indication*, sent from TCAP to the TC user. There are many types of primitives, but this thesis will only describe the dialogue-related primitives**.**

TCAP uses only SCCP's connectionless services, which means that TCAP does not set up persistent connections. Therefore, the intermediate layers between TCAP and SCCP (Presentation, Session, and Transport layer according to the OSI standard) that holds connections are completely transparent.

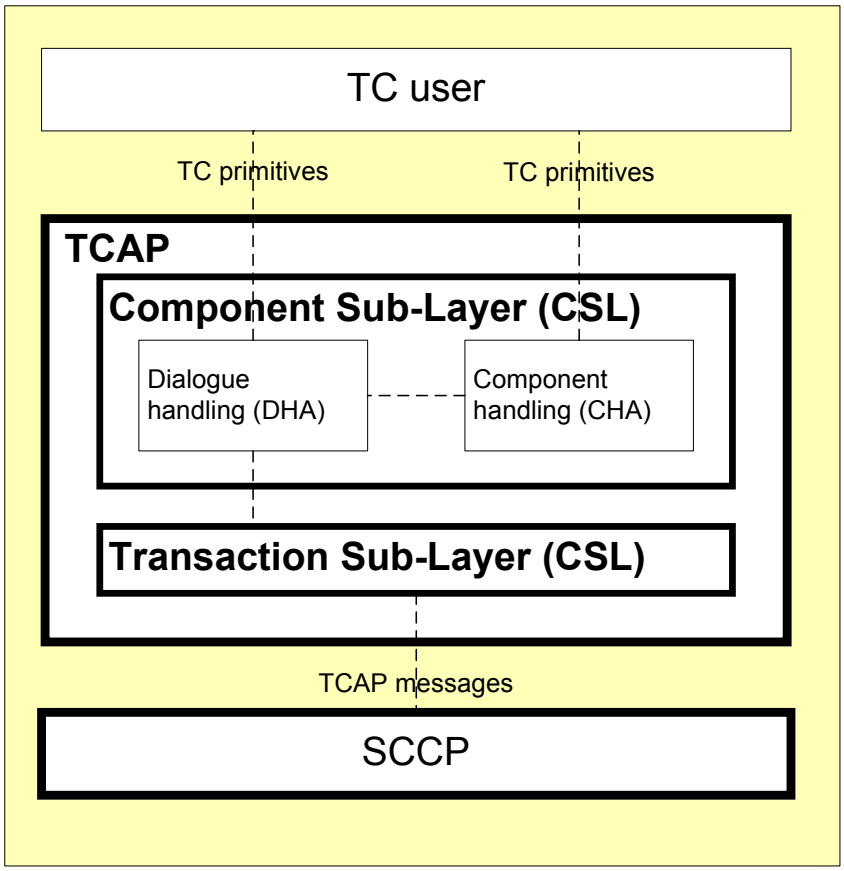


Figure 2.6: The internal structure of the TCAP layer [4].

Functionally, TCAP can be divided into two parts or so called sub-layers: the *Component Sub-Layer* (CSL) and the *Transaction Sub-Layer* (TSL), shown in Figure 2.6 above. The CSL handles the interface between the TC user and TCAP and the TSL handles the interface between SCCP and TCAP. The CSL provides the functionality to keep multiple dialogues with several operations active simultaneously. The CSL also enables the TC user to send and receive components (operation invokes or results).

In the CSL, two facilities are provided: *Dialogue Handling* (DHA) and *Component Handling* (CHA).

The DHA handles dialogues by offering two types of dialogues to send the information:

- Structured dialogues are more suitable for transferring larger amounts of data, as in requests and results. The simplified lifecycle of a dialogue can be described by setting up a dialogue, sending information in the dialogue, and terminating the dialogue. The

structured dialogues also make it possible to have several parallel dialogues active simultaneously.

- Unstructured dialogues are more suitable for sending operations that do not require replies, as it does not do initiations, results or terminations. These dialogues are terminated as soon as the component is sent.

The CHA handles the requests to perform operations by tagging the operation with an Invoke ID. This makes it possible to have several invocations active simultaneously.

The TSL handles the connectionless communication of sending TCAP messages from one TCAP node to another and tags the messages with a Transaction ID.

TCAP handles several types of primitives when communicating with the SCCP or the TC user. In this thesis, we will only make use of "Structured Dialogue Primitives" between the TC user and the Dialogue Handling sub-function mentioned above. This group of primitives consists of three types of primitives:

- **TC_BEGIN:** This initiates a dialogue with the recipient.
- **TC_CONTINUE:** This primitive is used after a dialogue has been initiated and before the same dialogue is terminated.
- **TC_END:** Any side of the connection can send this primitive to terminate the dialogue.

Common for all three dialogues is that they are all identified by a dialogue id, and they can be either a request or an indication. The dialogue id associates the primitive with a specific dialogue and distinguishes them from other ongoing dialogues.

Mobile Application Part (MAP)

The Mobile Application Part (MAP) [4, 7] is placed in the application layer together with TCAP and uses TCAP for its dialogue and component capabilities. The protocol also uses SCCP for its connectionless signaling services to perform peer-to-peer signaling between MAP enabled nodes in a PLMN (Public Land Mobile Network). A PLMN is a network

(owned by a single operator) that provides basic and extended telecommunication services for mobile terminals, such as cellular phones.

MAP was specifically designed to meet the signaling requirements of the *Global System for Mobile Communication* (GSM). A few examples of these requirements are for handling *Short Message* (SM) forwarding for *Short Message Service* (SMS), and subscriber data management. Some of the main services include location registration for *Mobile Stations* (MSs, defined below), handling and management for subscriber information, "handovers", and transferring of security data. Location registration is possible by keeping *Visitor Location Registers* (VLRs) and Home Location Registers (HLRs) up to date as the MS is roaming between different networks (HLR and VLR described below). Subscriber information is stored for service and billing purposes. A "handover" refers to the process of seamlessly transferring the MS's radio connection from one radio base station to another during an ongoing call, without breaking the call connection. The differences between the handover procedure and roaming is that the MS is idle or free for calls during roaming and the handover requires additional amounts of signaling to keep the radio and call connection of the MS.

The VLR and HLR mentioned above are databases, or *Application Entities* (AEs) in MAP, storing the location information of subscribers in the network. See Figure 2.7. The VLR stores the visiting subscriber's location as soon as it comes within its network, and then notifies the subscriber's HLR of its location. When the subscriber is called, its HLR is queried first to find out in which network the subscriber currently resides. An MS, as mentioned above, is a mobile terminal that the subscriber uses to communicate in the network, for example a cellular phone.

The *Mobile services Switching Center* (MSC) [4, 5] is the central component in a mobile network subsystem. Figure 2.7 illustrates a simplified and limited version of this subsystem. This center is responsible for routing all signals related to call processing. The MSC is interfaced with the *Base Station Controllers* (BSCs, which maintains the radio connection with MSs), the HLR, the VLR, any external networks (fixed or mobile) and more that we will not describe here, as it is not within the scope of this thesis. The MSC, in combination with the HLR and VLR, handles the call routing and roaming procedures.

For routing Short Messages (SMs), the MSC interfaces two additional *network entities* (NEs):

- The *Short Message Service Interworking MSC* (SMS-IWMSC) [5, 17] is placed in the same network as the MSC. This center receives and processes the SMs that come from the same network as the MSC (messages that originates from the MSC's network).

- The *Short Message Service Center* (SMSC [5, 17] that the MSC interfaces is located in a remote network. This center forwards the messages from an external network to the MSC, that is, messages that originates from an external network.

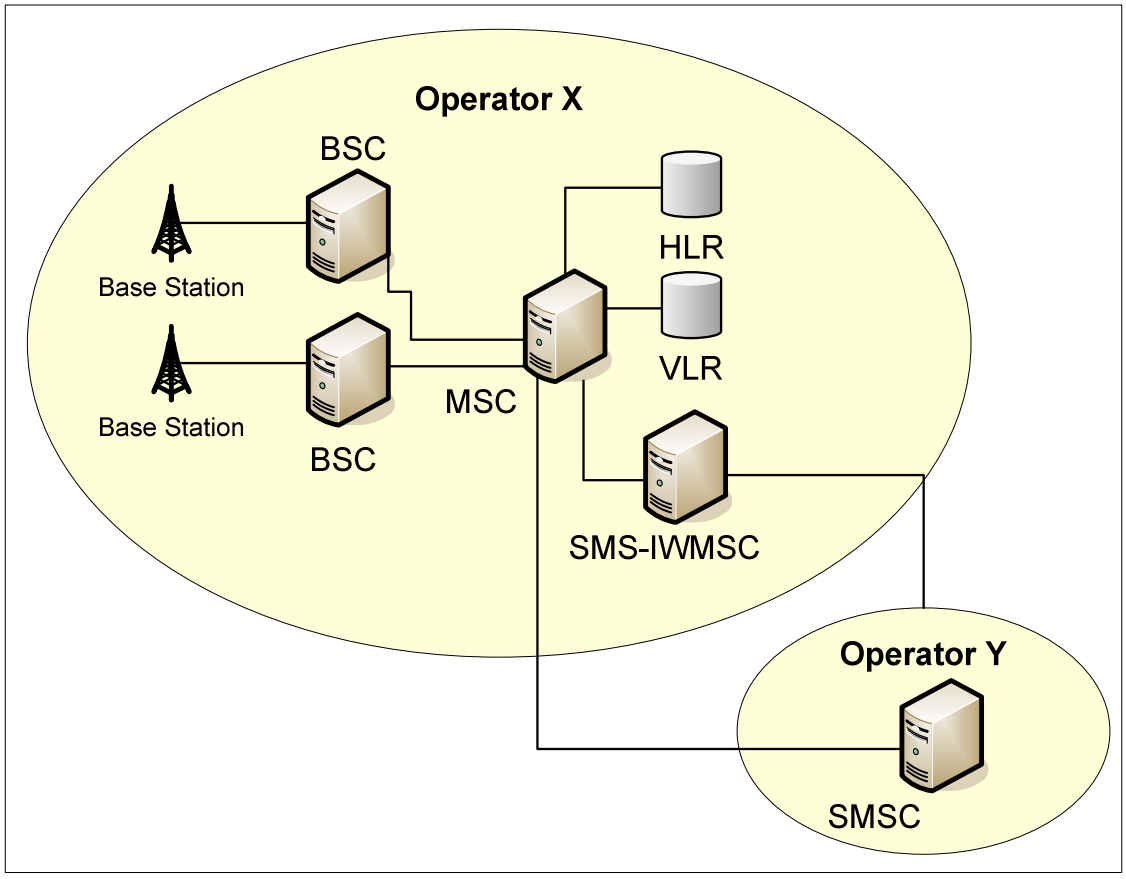


Figure 2.7: A simplified illustration of a mobile network subsystem.

2.4 Network Security

Today, network security [18] is a hot topic. More and more sensitive information is being communicated, which in turn attracts an increasing number of fraudulent actors and more

complex exploits of the network. These actors attack the network to gain valuable information or to invoke disruptive operations. The attacks can be classified as either passive attacks or active attacks.

The passive attack's purpose is to gain sensitive information and do not necessarily disrupt the ongoing communication. A couple of examples of passive attacks are:

- *Eavesdropping* on the communication traffic to gain sensitive or confidential information.

- *Analyzing the traffic* to gain information such as location and identity of the communicating parts and also the nature of the communication. Often applied when the traffic is encrypted or somehow scrambled.

Active attacks are meant to cause changes in the communication by modifying existing traffic or creating their own fraudulent traffic. Some active attacks are:

- *Masquerading* by impersonating a valid originator and making the destination believe that the traffic is legitimate.

- *Replaying* by capturing a pre-authorized and properly encrypted message and then resending it later to the receiver, making the receiver believe that the message or invocation is legitimate.

- *Modification of the message* by taking a legitimate message and making changes in it to cause harmful or exploitative effects on the receiver. The changes are done without disrupting the message too much so that it still seems legitimate.

A few types of security services have been created to prevent both of these kinds of attacks. Some examples of types of services are *data confidentiality*, *data integrity*, *data origin authorization*, and *replay protection*.

Data confidentiality

Data confidentiality is used to hide the contents of the message so that a third party cannot read the contents. This confidentiality is achieved by encrypting the data that is to be sent. Encryption is accomplished by the use of complex mathematical algorithms and relatively

large encryption keys. The keys should be 56 bits or longer since a shorter key is considered to be too easy to crack. Two different types of encryption algorithms can be used: *symmetric* or *asymmetric* encryption algorithms.

The main difference between these two types of algorithms is that the symmetric kind uses the same key to encrypt and decrypt the message while the asymmetric encryption uses key-pairs. The key-pairs are designed in a way that if a text or message is encrypted with one key, then it can only be decrypted with the other corresponding key in the pair. This type of encryption algorithm is more complex and has a major advantage over symmetric algorithms, namely key distribution. The key-pair consists of one *public key*, accessible by the general public, and a *private key*, accessible only to the entity that generated the keys. The public key can be distributed freely without compromising the security. A drawback of asymmetric algorithms is the computational overhead of generating the keys. Asymmetric encryption will not be described further in this thesis, as we will only use the symmetric algorithms specified in [19].

The symmetric algorithms can be publicly known but the keys must be kept secret to preserve confidentiality. This secrecy requirement introduces some requirements on the algorithm. The algorithm must be strong enough so that an adversary cannot decipher the data by only having knowledge of the algorithm. The algorithm must also be so strong that an adversary cannot calculate the key (within reasonable time) by knowing only the ciphertext[2] and the algorithm. Another requirement (or problem) is that both the sender and the receiver must have the key and store it securely so that no outsider can have access to it. Anyone who attains access to the key can read all the messages that have been encrypted with the (no longer secret) key.

In this thesis, we will be using the *Advanced Encryption Standard* (AES) algorithm in counter mode with a 128-bit key. For more information on the AES algorithm, see [18, 20]. The counter mode is described in [21].

Data origin authentication & data integrity

Origin authentication mechanisms are needed to verify that the two communicating nodes are who they claim themselves to be. This service, or mechanism, is applied mainly to avoid the case of a masquerading attack from an unauthorized node. Authenticating nodes is critical for

[2] A ciphertext [18] is the result of a plaintext that has been encrypted and is unintelligible. The ciphertext is dependent on the plaintext message and the encryption key.

handling charging information. There are several ways to implement origin authentication without encryption of the message. One example algorithm is the *Message Authentication Code* (MAC) [18]. This algorithm calculates a checksum, the MAC, based on the message and a mutually known secret key. The checksum is then appended to the message. The receiver of the message performs the same calculation and compares the result with the appended MAC. A positive comparison confirms an authorized origin of the message.

The fact that the MAC is dependent on the message and a secret key makes it similar to encryption. The main difference is that the algorithm for MAC does not need to be reversible, unlike the encryption algorithm.

Data integrity mechanisms are used to verify that the message contents have not been changed since the message left the originating node. Data integrity is dependent on origin authentication**.** Without authenticating the origin, the integrity mechanisms would only be able to protect against malicious modifications, but not against malicious data sent directly from the arbitrary and unauthenticated origin. Data integrity algorithms are based on encryption algorithms to calculate and encrypt the checksum of the message in a similar fashion as MAC, mentioned above. In this thesis we will use the AES algorithm in a so-called *Cipher Block Chaining* (CBC) [18] mode. For a description of this mode, see [22].

Replay protection

Replay protection is used to prevent a third party from taking a properly authenticated, encrypted and integrity calculated message from the traffic stream and resending the message at a later, potentially disruptive, time to invoke unauthorized operations. To prevent this destructive behaviour, the message is tagged with a timestamp. The timestamp is included into the content that goes through the integrity calculations, which make the timestamp hard to change without the message failing the integrity check. If the receiver receives a message with an old timestamp, it can simply discard the message without any harm done.

2.5 Motivation

In the few months prior to the creation of [23] in June 2004, there was a rise in the number of SMS frauds (described in [23]). There are many kinds of fraudulent SMS usage, but we will only describe one scenario here, namely SMS spamming.

To understand the procedure of SMS spamming, one must understand the inter-operator communication routines. The process of sending a Short Message (SM) between two mobile operators using MAP is done in two steps:

1. The Short Message Service Center (SMSC) that receives the SM from its own network sends a routing information request to the HLR (see Section 2.3.2) of the receiving MS. The request is a MAP message called "sendRoutingInfoForSM" and contains a *Mobile Station Integrated Service Digital Network* (MSISDN) number, basically a mobile telephone number. Given that the MSISDN number is a valid subscriber's number, the HLR replies with the receiving MS's *International Mobile Station Identity* (IMSI) and the currently valid MSC address whose network the MS currently resides in. Figure 2.8 illustrates this step.

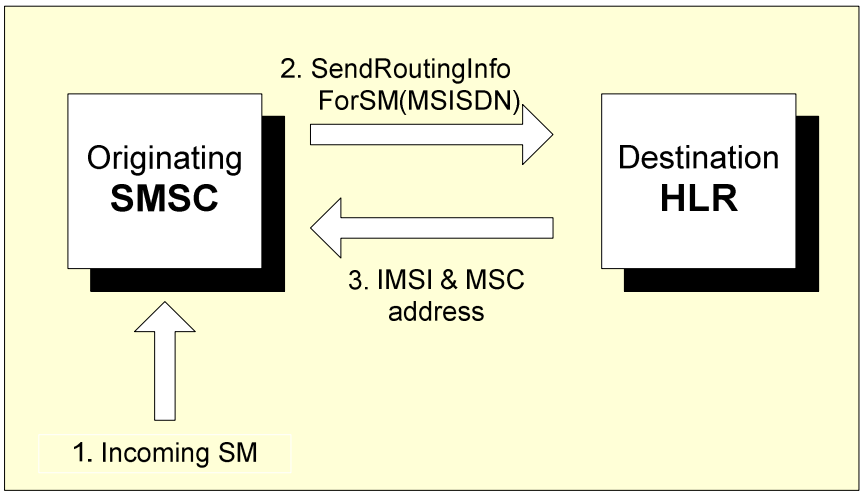


Figure 2.8: Requesting routing information for a short message (SM).

2. After receiving the reply, the SMSC has the necessary information to send a SM directly to the MSC that the MS resides in. To send the SM, the SMSC uses a MAP message called Forward Short Message (mt-forwardSM). The receiving MSC acknowledges the message and generates the relevant information to store, which includes charging information and the address of the originating SMSC. See Figure 2.9.

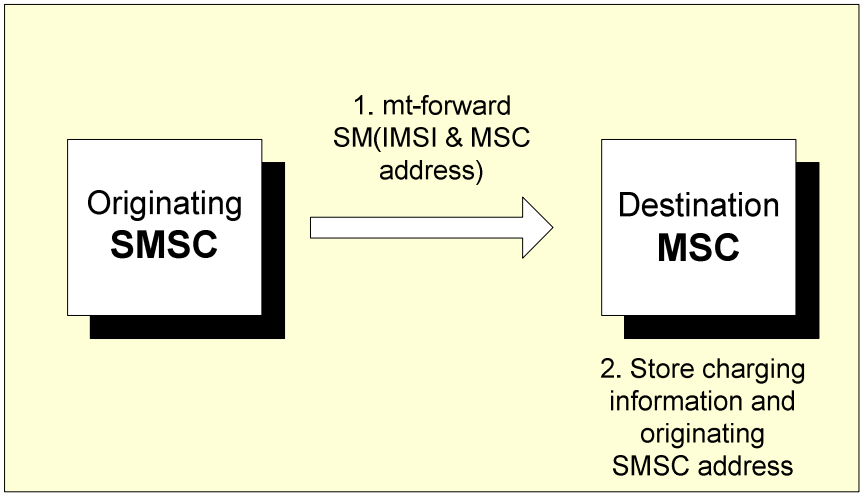


Figure 2.9: Sending a short message (SM) and registering charging information.

It is important to know that these two steps are not in any way bound to each other. That means that the steps can be performed completely independent of each other in an unregulated number of times.

The result of this independence is that it is possible to create a large database containing MSISDN, IMSI and MSC address numbers by performing step 1 (Figure 2.8) over a large range of MSISDN numbers. The information will also be highly accurate as the information is taken directly from the recipients HLR and all the invalid MSISDN numbers will be sorted out by the HLR.

In the scenario of SMS spamming, step 1 is performed repeatedly to create the database. After acquiring the information, step 2 (Figure 2.9) is performed by a fraudulent SMSC that starts sending out mt-forwardSMs containing a fake originating SMSC address. The receiving MSC receives the message and simply forwards it onwards to the MSs.

A side effect of this fraudulent behaviour is that the charging information will be incorrect as the MSC stores the information based on the faked SMSC address, resulting in that the operator of the faked SMSC address will be charged incorrectly. See Figure 2.10.

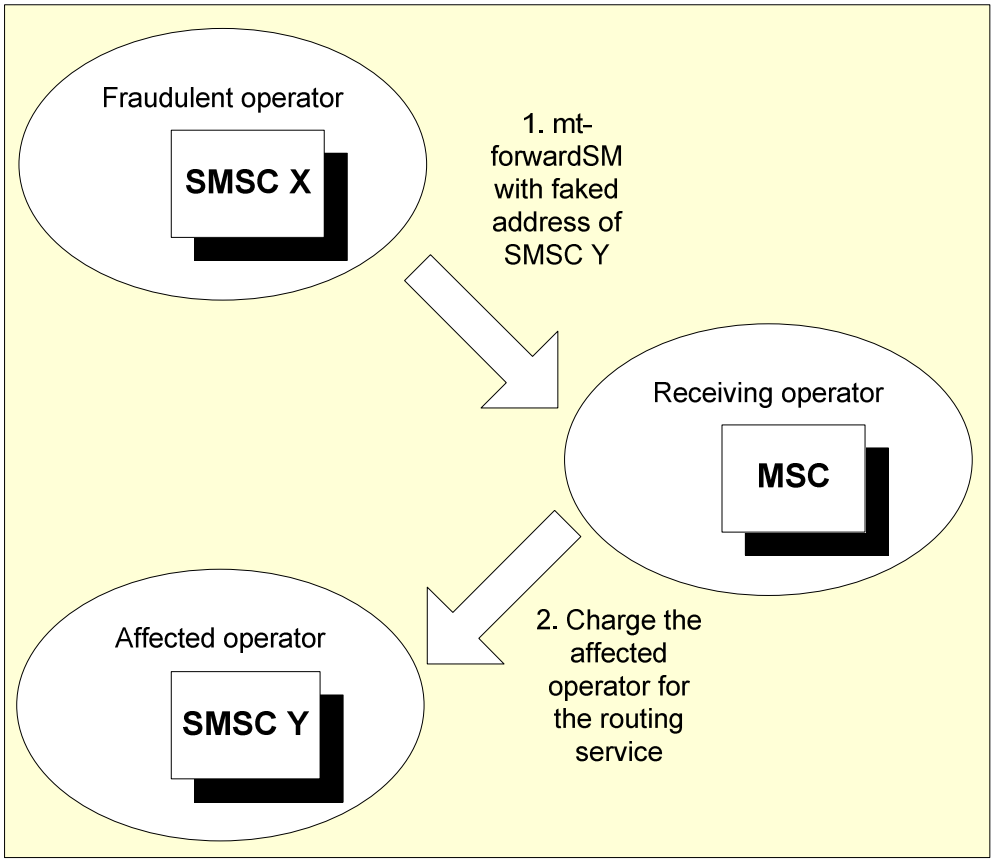


Figure 2.10: A SMS spamming scenario.

The affected operators are eager to find a solution to this problem. SS7 was designed based on complete trust between all operators and due to this mindset when designing SS7, there is nothing preventing anyone with access to the international SS7 network from sending fraudulent signaling messages.

In [23] three alternative solutions were presented to counteract these kinds of fraudulent acts.

1. SS7 over IP uses the *Signaling Transport* (SIGTRAN) protocol together with security mechanisms based on IP networks that was specifically designed for handling IP traffic (described in [24]). This method is by far superior as it provides end-to-end protection for all the information from, in this case, the MAP message payload to the SCCP addresses.

2. *Mobile Application Part security* (MAPsec) (described in [25]) protects specific groups of MAP messages (sendRoutingInfoForSM and mt-forwardSM not included)

with specified protection profiles. This solution requires the use of a central *Key Administration Center* (KAC) that interfaces all the MAP nodes with a protocol called the "Ze-interface" to securely distribute secret keys to all MAP nodes.

3. The third alternative is to create a correlation between sendRoutingInfoForSM and mt-forwardSM by using a token with a limited lifetime. The token is returned after the sendRoutingInfoForSM request and replaces the MSC address the SMSC receives. This token must be used when sending an mt-forwardSM message.

The first alternative is unlikely to be realized anytime soon, as SIGTRAN is not yet standardized in the *Internet Engineering Task Force* (IETF, an international standardization organization for these kinds of protocols). Also, the majority of the operators are not yet willing to install SIGTRAN and this will probably not change in the near future.

The second alternative had not yet been completely standardized (as the document at the time was an older version [26]) and, as mentioned, did not handle the specific type of SMS fraud described above. Also, the "Ze-interface" was and still is far from complete. Completing the interface would require a major workload.

The third alternative was of little interest, as it did not protect against the described scenario of SMS fraud and also required major changes in a large number of NEs. The token allocation, which is a major part of the solution, was never standardized.

Given the three alternatives, the MAPsec solution seemed the most realistic. The "Ze-interface" was not yet standardized, but it would be possible to create "MAPsec gateways" with a limited KAC functionality that would protect the inter-operator communication and it may be added later to the SIGTRAN solution.

Since the creation of [23], the specification on MAPsec has evolved, and a new specification called TCAP user security (TCAPsec) [19] has emerged that replaces MAPsec. This new specification is what the authors are supposed to build a prototype of.

Another solution worth mentioning is the TCAP handshake described in [27] and later commented in [28]. This solution was designed to counteract the specific scenario illustrated in Figure 2.10. The dialogue in an unprotected transaction of SM is illustrated in Figure 2.11 below.

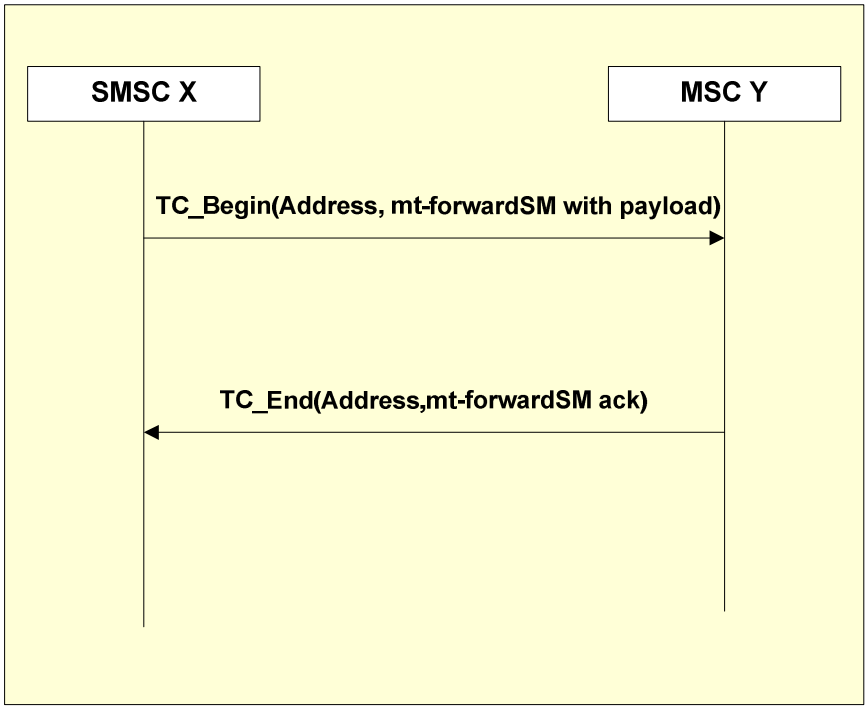


Figure 2.11: Forwarding SM data in an unprotected dialogue.

The idea behind the TCAP handshake is to force the sender to establish a dialogue before sending any kind of SM information. This is done by requiring the sender to first send an empty TC_Begin request message to the recipient and having the recipient send a reply on that request. It is only after this dialogue initiation that the sending of the SM can begin. See Figure 2.12.

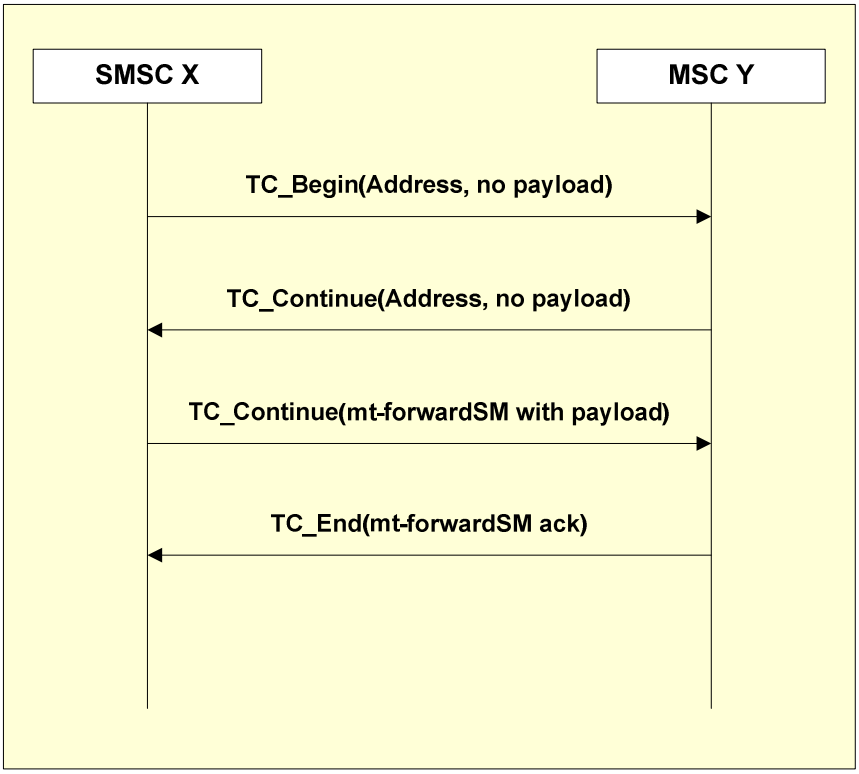


Figure 2.12: The TCAP handshake.

Establishing this dialogue guarantees that the originator is using the correct originator address. Therefore, this should counteract the fraud of spoofing[3] the originating address.

The TCAP handshake is a fairly simple procedure to implement, as all the required functionalities already exist in the current MAP and TCAP implementations.

Although this may seem like a good solution, there are still a few problems that need to be considered:

- The TCAP handshake effectively increases the traffic in the network with the initial dialogue setup.
- In [28], it is described how a fraudulent sender may circumvent this security mechanism by spoofing an originator address, predicting the transaction id, and sending the message. Upon arrival of the TC_Begin, the recipient will send a TC_Continue to the spoofed address. The fraudulent sender can then attempt to predict the transaction id

[3] Spoofing refers to the act of deceiving the recipient by inserting a falsified originator address in the message.

sent to the spoofed address. Using the transaction id, the fraudster can send the message with the payload and carry out an mt-forwardSM request. This must be done within a limited time window as the spoofed address node will detect the erroneous dialogue and send an abort for the dialogue. See Figure 2.13.

- The protection the TCAP handshake offers is severely limited as it is designed only to prevent the specific fraud scenario described above.

Given the simplicity of standardizing this solution, it is a good fast-to-deploy alternative for protection. However, with the problems mentioned above, TCAP handshake can only be a temporary solution until a better and more comprehensive solution is implemented to replace it (such as TCAPsec).

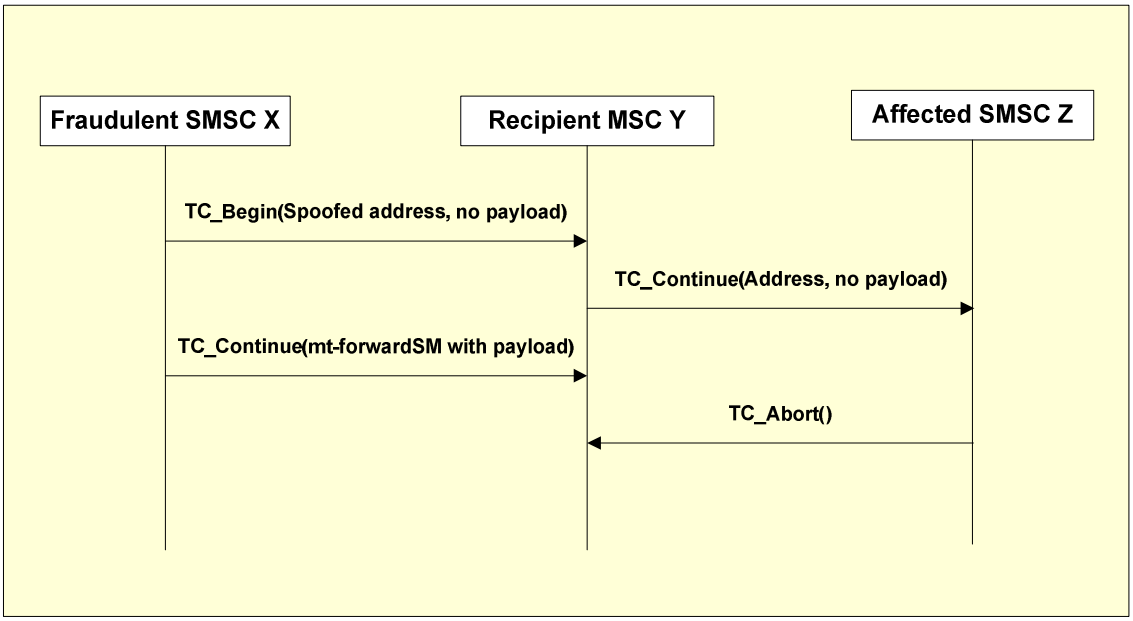


Figure 2.13: Circumventing the TCAP handshake protection.

2.6 Summary

In this chapter, some background information was presented to help understand the TCAPsec concept and the prototype design. The history and fundamentals of telecommunication and signaling has been briefly described. Furthermore, the structure of the SS7 signaling stack has been explained and compared to the OSI and TCP/IP stack. Each layer in the SS7 stack that is related to the TCAPsec concept has been presented and described. Some of the essential aspects of network security have been presented to give an overview of the subject. Finally, the motivations and problems that led to the creation of TCAPsec have been described to facilitate the understanding of the TCAPsec concept. Also, some alternative solutions were presented.

3 Description of TCAP user security (TCAPsec)

This chapter describes the TCAPsec concept based on the TCAP user security technical specification [19]. A brief overview of the concept will be given in Section 3.1. Then a more detailed description will be presented in Section 3.2, where all the related entities will be identified and described in detail. In Section 3.2.1 the function of the databases will be described together with the contained attributes. In Section 3.2.2 the message structure of the different protection modes are explained and the process of protecting messages is illustrated and summarized.

3.1 Overview

The TCAP user security (TCAPsec) [19] concept was initially created to provide signaling traffic security between security domains to all TCAP users, independently of application protocol type. A security domain is defined as a PLMN (described in Section 2.3.2) or an operator network. A TCAP user is defined as an entity that is assigned an SSN (see SCCP in Section 2.3.2). The basic protection requirements consist of confidentiality, integrity, authentication (for both origin and entity), and anti-replay protection. These services are provided using cryptographic techniques. The set of all the enhancements and extensions used to provide protection for TCAP messages is called TCAPsec.

The TCAPsec specification is based on a gateway concept. This concept implies that all traffic, inbound or outbound, from a security domain has to traverse an *SS7 Security Gateway* (SS7 SEG) for inter-operator or inter-PLMN protection. The SS7 SEG then inserts or removes protection of the messages based on pre-negotiated policies regarding other operators or PLMNs.

3.2 TCAPsec - detailed description

TCAPsec defines security mechanisms used for protecting all TCAP users from inter-operator active attacks. An alternative to TCAPsec is NDS/IP described in [24] which is used for IP-based networks.

TCAPsec requires at least one SS7 SEG in each intercommunicating PLMN. These SEGs will be located at the border of the PLMN for protection of the inbound and outbound inter-domain messages. Each of these SEGs will contain policy information known to both sides. The policy information is important in order to synchronize the level of protection for both sides and must be negotiated and set before any messages are sent between the operators. Also, a number of *Security Associations* (SAs), known only to the two communicating operators, must be negotiated before any message transaction can begin. An SA contains a set of parameters required to initiate and maintain a secure connection between two SS7 SEGs.

For routing purposes, the protected traffic will be undistinguishable to all NEs except the sending and receiving SEGs.

A protected network has one or more SS7 SEGs on the border of the network for the incoming and outgoing messages to traverse. The SS7 SEGs will provide protection for the communication channel between the PLMNs, as illustrated in Figure 3.1 below. A SEG can be co-located with another TCAP user, but [19] only describes the case of stand-alone gateways.

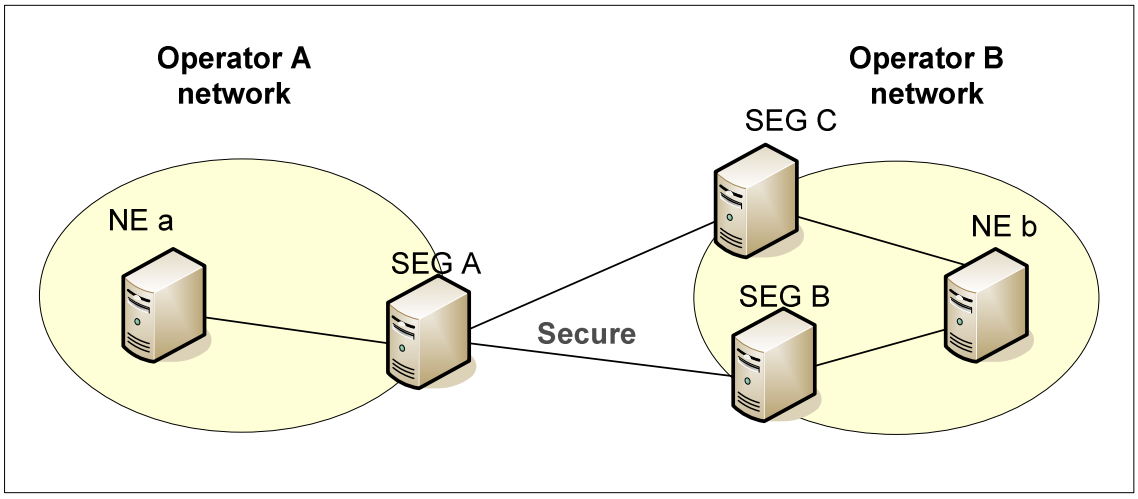


Figure 3.1: SS7 SEGs location at the border between operator networks.

3.2.1 Databases and attributes

The SS7 SEG contains or has access to two databases for storing and accessing the information required to establish a secure connection, illustrated in Figure 3.2 below. The first database is the *Security Policy Database* (SPD). The SPD contains one entry for each of the PLMN where communication is allowed. The entries in the SPD define the mode of protection to use towards one remote PLMN and must be consistent in all SS7 SEGs in both its own PLMN and the corresponding SS7 SEGs of the remote PLMN. Another attribute in the entry is the parameter called “fallback to unprotected mode”. This parameter shows whether or not the secure connection is allowed to fall back to unprotected mode, in case of a failed security mechanism. The use of this parameter is shown in Appendix C. TCAPsec can only be fully secure if this parameter is disabled.

The use of policies prevents the act of spoofing by requiring the incoming message to have a certain mode of protection, by using the pre-negotiated SAs. If an incoming message does not fulfil the negotiated policy, the message is dropped. The policies are explicitly configured as to one policy per communicating PLMN. This way of configuration implies that if a remote PLMN that wishes to communicate does not have an entry in the SPD, its messages are rejected.

The exact contents and attributes of the SPD have yet to be defined.

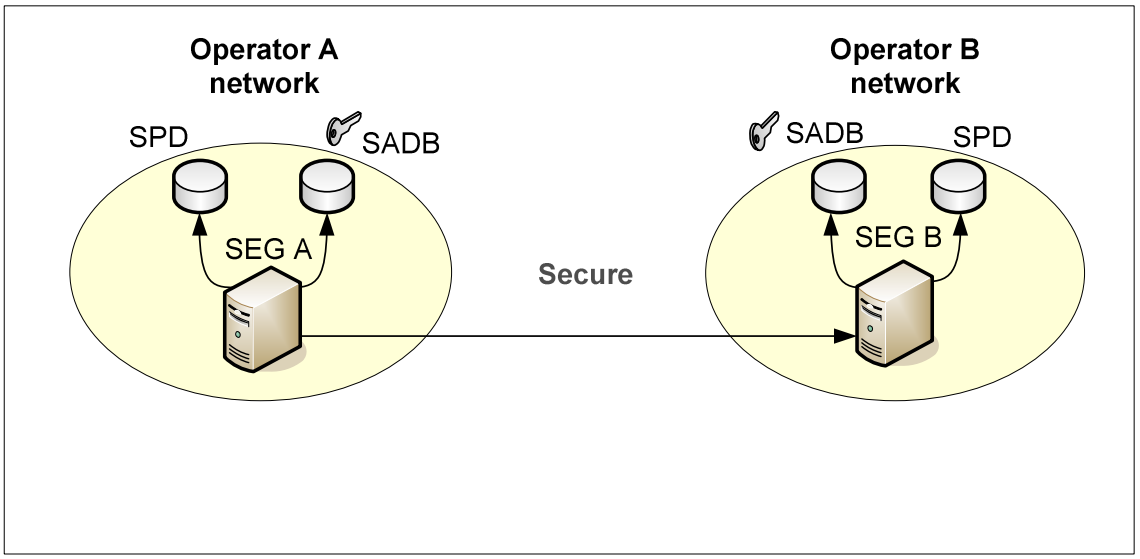


Figure 3.2: One SADB and one SPD for each SS7 SEG.

The second database is the *Security Associations Database* (SADB). This database contains attributes used for applying protection to the inbound and outbound messages. The policy entries in the SPD define a set of valid SAs used to communicate with a remote PLMN. Each SA is expendable and the SADB needs to be continuously or regularly refilled in order to preserve key freshness and avoid reuse of old keys in old SAs. If a key is reused, it will render a decreased level of protection against crackers, compared to using new keys. New SAs are negotiated via an external and centralized network entity called the Key Administration Center (KAC). This center has yet to be defined and standardized.

All the attributes in an SA are presented and described below:

- *Destination Network Id*: This is an identifier for the originating network of the message. The id consists of a *Country Code* (CC) and a *National Destination Code* (NDC). This attribute is used to identify the SA used to apply protection on an outbound message.

- *Security Parameter Index* (SPI): The SPI is an identifier used in combination with the Destination Network Id to identify the SA. The identified SA is in turn used for removing the protection on the inbound messages.

- *Sending Network Id*: The Sending Network Id has the same format as the Destination Network Id. This attribute identifies the originating network of the message.

- *SS7 SEG Encryption Algorithm identifier* (SEA): This attribute identifies an encryption algorithm from a predefined table of 16 entries (0-15). The mode of encryption is implicitly defined together with the algorithm identifier. So far only one encryption algorithm has been defined – AES in *counter* (CTR) mode with a 128-bit key and *Initiation Vector* (IV, for more information, see [18]). For more information on this algorithm and mode, see [20, 29]. The empty fields in the table are available for future encryption algorithm definitions and modes.

- *SS7 SEG Encryption Key identifier* (SEK): This attribute contains the key for the encryption algorithm.

- *SS7 SEG Integrity Algorithm identifier* (SIA): The SIA identifies an integrity algorithm from a predefined table of 16 entries (0-15). Similarly to the SEA, the encryption mode is implicitly defined in the algorithm identifier. This table also only has one entry defined so far: AES in CBC MAC-M mode with a 128-bit key, without IV. This mode of block cipher is described in [22]. MAC-M refers to Message Authentication Code used for TCAP users. The empty fields in the table are available for future integrity algorithm definitions and modes.

- *SS7 SEG Integrity Key identifier* (SIK): This attribute contains the key for the integrity algorithm.

- *SA hard expiry time*: The hard expiry time defines the actual expiry time of the SA.

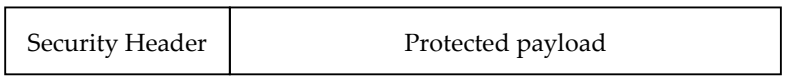
- *SA soft expiry time*: The soft expiry time defines the expiry time of using the SA on outgoing messages.

After the hard expiry time has passed, the SA can no longer be used for any kind of traffic, inbound or outbound. After the soft expiry time has passed, the SA will not be used for protecting outbound messages, but still for inbound messages, unless all stored SAs in the SADB are expired.

3.2.2 Protection modes

TCAPsec offers three modes of protection. These protection modes provide three different levels of protection for the messages by using cryptography.

The protected payload or message will have a security header prepended to the message. The content of the header depends on the protection mode and is always in plaintext.



The header can contain a number of attributes described below:

- *Security Parameter Index* (SPI): Described above.
- *Original component Id*: This attribute identifies the type of TCAP primitive in the protected payload, such as Invoke-, Result-, or Error-requests (see TCAP in Section 2.3.2). (This attribute was removed completely in version 1.0.0 of the technical specification of TCAP user security released in November 2005.)
- *Time Variant Parameter* (TVP): This is a 32-bit timestamp used for anti-replay protection. The messages must arrive at the receiver within a defined time-window to be valid. Otherwise the message is deemed old and is dropped.
- *SS7 SEG Id*: A unique identifier used to create an IV.
- *Proprietary Field* (Prop): This field is used to create an IV. The exact use of this field has yet to be standardized.

Common for all the protection modes is that the receiving SS7 SEG checks the TVP on the message and ensures that the message has the right mode of protection according to set policy. If the TVP is too old or if the message has the wrong protection mode, the message is simply dropped.

The three modes of protection that TCAPsec offers are the following:

- **Protection mode 0:** The lowest level of protection offers no protection at all.
- **Protection mode 1:** The next level of protection offers integrity and authentication services by the use of MACs.
- **Protection mode 2:** The highest level of protection offers the same security services as mode 1 with the addition of confidentiality by performing total encryption of the message.

Protection mode 0

The security header of the messages in protection mode 0 is very simple and contains only the SPI and Original component Id.

$$\textit{Security header} = \textit{SPI} \,||\, \textit{Original component Id}$$

The messages using protection mode 0 security will only have a security header prepended to it. (In version 1.0.0 of the technical specification [17], the Original component Id was removed from the header and replaced by a TVP tag for anti-replay protection.)

Protection mode 1

Protection mode 1 provides integrity and authentication services. The security header for this mode contains the SPI, Original component id (removed in version 1.0.0 of the TCAP user security technical specification [17]), TVP, SS7 SEG id and a proprietary field.

$$\textit{Security header} = \textit{SPI} \,||\, \textit{Original component Id} \,||\, \textit{TVP} \,||\, \textit{SS7 SEG Id} \,||\, \textit{Prop}$$

Integrity and authentication is achieved by processing the security header and the message with the integrity algorithm defined by SIA in the sending SS7 SEG. After the algorithm has been executed, the resulting last 32 bits (MAC-M) is appended to the end of the plaintext message before sending it.

Security Header	Plaintext || MAC-M(Security Header || Plaintext)

The receiving SS7 SEG will then extract the MAC-M from the message, perform the same integrity calculation on the message and security header, and compare the result with the extracted MAC-M. A positive comparison confirms that the message is unchanged and that the message originator really is whoever it claims to be, as the SIK is secret to all but the communicating operators. This way the integrity can be confirmed and the authentication is confirmed by usage of the correct SIK.

Protection mode 2

Protection mode 2 is the most secure mode. This mode introduces confidentiality to the message. The security header in this mode is identical to the header in mode 1. The first step in applying protection in this mode is to perform encryption on the message with the encryption algorithm in SEA, the key in SEK, and the IV. This procedure provides the message with full confidentiality. The second step is to perform the integrity calculation on the security header and the ciphertext. The resulting MAC-M is then appended to the ciphertext before forwarding the protected message.

Security Header	Ciphertext || MAC-M(Security Header || Ciphertext)

On the receiving end, the process is reversed. The first step is to extract the MAC-M and perform the integrity calculation on the security header and ciphertext. The next step is to perform the decryption on the ciphertext before forwarding the message to the end node.

The IV is created using the previously described parameters in a sequential form:

$$\textit{Initiation Vector} = \textit{TVP} \,||\, \textit{SS7 SEG Id} \,||\, \textit{Prop} \,||\, \textit{Pad}$$

The padding is appended to get the right size of the vector. The padding consists of a string of zeros.

3.2.3 Message protection procedure

The complete flow of inbound and outbound message is illustrated in two flowcharts in Appendix C.

The process of protecting a message can be described in six very simplified steps. These steps are illustrated in Figure 3.3. This description begins when an outgoing, unprotected message arrives at SEG A, and ends when SEG B has processed the message and its protection has been removed.

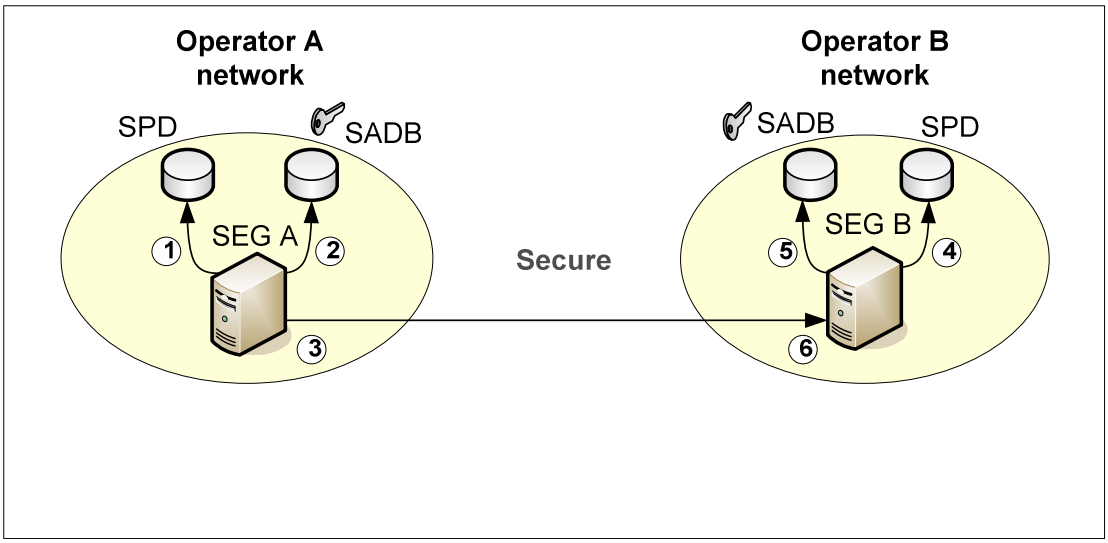


Figure 3.3: The process of protecting an inter-operator message.

1. The destination network address of the message is extracted and compared to the entries in the SPD.

2. Using the security policy read from the SPD, the SADB is checked for a valid SA.

3. The unprotected message is processed with the protection mode defined in the policy using the obtained SA. The protected message is then forwarded towards the SS7 SEG of the destination network, in this case SEG B.

4. The protected message is received at SEG B. The SPD is checked to verify the security policy associated with the originating network.

5. With the policy verified, the SADB is checked to retrieve the SA identified by the SPI in the security header of the protected message.

6. The protected message is processed and unprotected in the protection mode defined by the policy from SPD. The process uses the SA retrieved from the SADB to remove the protection. Finally, the TVP of the message is checked for anti-replay protection. At this point, the message is unprotected and ready to be forwarded to its final destination within the PLMN.

3.3 Summary

This chapter has presented the fundamentals of the TCAPsec concept. An overview of the concept was presented together with the offered security services. Furthermore, the gateway concept in TCAPsec was explained together with the contents of the security gateways (SEGs). The functionality of the databases in the SEGs and the contained attributes were described in detail. The policy database in the SEGs contains information on the mode of protection used, and the security association database contains attributes used for protecting the packages. The protection services in form of three protection modes were specified and the procedure of protecting messages was explained.

4 Design and prototyping

This chapter states the functional requirements and the design of the TCAPsec prototype. The chapter begins by describing the design method used for designing the prototype in Section 4.1. The scope and the limitations made to the prototype are defined in Section 4.2. Section 4.3 identifies the functional requirements that the prototype has to fulfil. The actual design of the prototype is described in Section 4.4, with an overview of the software and hardware architecture in Section 4.4.1 and a detailed description of each node in Section 4.4.2. Section 4.5 states a number of testing requirements for testing the prototype functionality. The prototype testing results are presented in Section 4.6, with the discovered limitations described in Section 4.6.1.

4.1 Design method

It was decided that the design process would not strictly follow any known modeling or design methodologies such as the *Unified Process* (UP) [30] or *Extreme Programming* (XP) [31]. This decision was based on the assumption that the authors did not have enough experience in handling design methodologies. The lack of experience in turn implied that it would be a risk to take the time needed to learn the methodology, as the needed time could get unreasonably long. Based on this assumption, the authors decided to make the design without the use of UML or XP. Instead, the improvised design process was similar to the waterfall method and consisted of four steps.

1. Investigate and document the requirements on the prototype. This was done by studying the technical specification for TCAPsec and identifying the required entities and functionalities for creating a prototype.

2. Document the testing procedure requirements for the prototype. With the entities and functionalities identified, testing procedures were defined to test the functionality and stability of the prototype implementation.

3. Create the design on the prototype based on the documented requirements. The design had to follow the functional requirements to pass the defined tests.

4. Implement and test the prototype iteratively according to the design and documentation.

After the prototype was completed, the performance tests could finally be performed.

4.2 Scope

When we first begun our work, the thought was to perform a comprehensive study over the performance costs of introducing TCAPsec. However, after studying the given material we realized that the task was too extensive to be performed in 20 weeks. When we got a clear understanding of the task, we started to limit the included parts.

The main purpose of the prototype is to be a platform for measuring the performance costs of routing and integrity calculations and encryption of the routed messages, similar to the TCAPsec concept. Therefore, the scope of the prototype design is limited to the most basic and necessary functionality required to route and protect the messages. The simulated network will consist of a minimum of four nodes, simulating two or more operator networks. Each operator network must have at least one security gateway (SEG), interfacing the other networks. The SEGs will contain the required security mechanisms for protecting the messages.

One major part of the TCAPsec concept that was not included in our prototype design is the Security Association Database (SADB) and the Security Policy Database (SPD), described in Section 3.2.1. One of the reasons for the exclusion of the databases is the fact that the key management is not yet standardized. The keys in the SAs that are needed for security are hardcoded instead of collected from these databases.

4.3 Functional requirements

The following requirements need to be met for the prototype to actually perform the given task properly.

The prototype should consist of at least four nodes that are interconnected in a serial connection (see Figure 4.1). All the nodes must have a SS7 stack that has at least TCAP capabilities. All messages will be sent as payloads in TCAP dialogues.

The nodes are divided into two types: end nodes and SEGs. The end node will represent any node in an operator's network that wishes to communicate with a node in a different operator's network. This type of communication between operator networks is called inter-operator communication. All traffic between end nodes will be routed through the SEGs at the border of the respective operator's network.

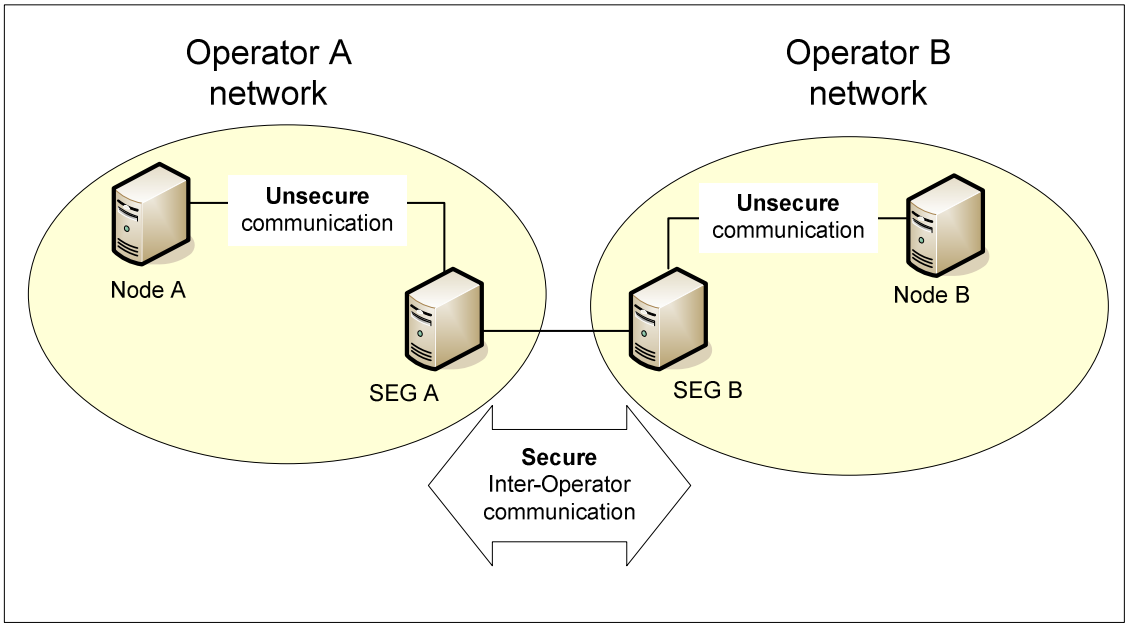


Figure 4.1: Four serial connected nodes with protected inter-operator communication using SEGs.

- The SEGs will represent nodes on the network border that is traversed by all inter-operator communication messages.
    - The SEGs will also be responsible for the insertion and removal of protection on the messages entering or leaving the operator's network. After the protection has been added or removed, the message is routed forward along its path to the remote end.
    - All the SEGs should store mutually known 128-bit secret keys and initiation vectors (IVs) used for authentication, integrity, and AES encryption/decryption.

- All the messages from the applications above TCAP, independent of protocol (for example MAP), should be protected in the inter-operator communication. This application-independent kind of protection means that the messages, or the payloads, in the TCAP dialogues should be protected (that is, when protection is supposed to be applied).

- There are three different modes of protection (described in Section 3.2.2) that the prototype is required to provide:

  - **Mode 0 – No protection**: In this mode there will not be kind of protection. The messages are to be routed between the end nodes and through the SEGs without any interference by the protection mechanisms.

  - **Mode 1 – Authentication & Integrity**: In this mode the messages will process the integrity algorithm on the payload and use the mutually known secret keys for authentication. The integrity algorithm in this prototype will be AES in CBC mode in combination with the secret keys. In the SEG of the originating network, the result of the integrity calculation, a 32-bit block, will be attached to the payload and delivered to the designated SEG. The receiving SEG will then take the received message and extract the 32-bit block, perform the algorithm with the same key, and compare the result with the attached block. A positive comparison indicates an authenticated originator and an unmodified message.

  - **Mode 2 – Encryption**: This mode will provide the highest level of protection. The message will be encrypted with the AES algorithm in counter mode and the mutually known key and IV. Additionally, the resulting ciphertext will go through the same authentication and integrity procedures as described above in Protection mode 1 before being forwarded. The receiving SEG then receives the message and does the same procedure in the reverse order. The result should be a plaintext message that has been decrypted, authenticated, and integrity checked.

The user should be able to switch between these three modes of protection by changing a single value. The SEGs will set the level of protection based on interpretation of this value.

- Other than providing secure inter-operator communication, the prototype also has to time the routing of the messages using a timer. This is to measure the routing time taken for routing in the different protection modes. The goal is to measure the time taken from when the message is sent from the end node in one operator's network (Node A in Figure 4.1), to the other end node of the remote operator's network (Node B in Figure 4.1) receives the message.

4.4 Prototype design

The prototype was designed to meet the requirements described in Section 4.3. The description of the prototype is divided into two parts to give a better overview of the design. The first part in Section 4.4.1 will give a brief overview of the solution by describing the nodes relative to the stacks and hardware. The second part in Section 4.4.2 will give a more detailed description of each node.

4.4.1 Prototype network architecture

Two servers, with one installed and preconfigured stack each, were provided by TietoEnator. The servers were installed with Sun Microsystems's operating system: Solaris 5.8. The SS7 protocol stacks installed were "Sol R4.0/SS7" or later. There are more details to the protocol stacks, but those will not be described here, as they are TietoEnator's proprietary solutions.

The prototypes were connected as applications to the TCAP layer in the stack, interfacing the TCAP *Application Programming Interface* (API) version ITU R5F/C. The Figure 4.2 shows this connection between application and stack.

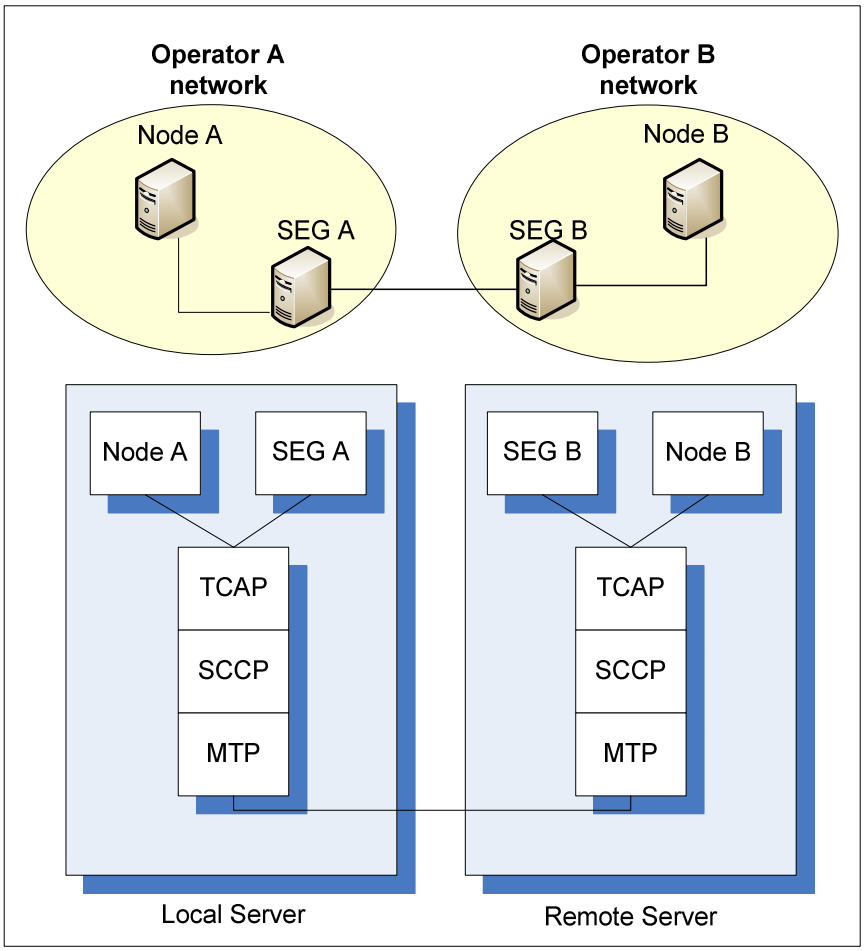


Figure 4.2: The structure of the implementation compared to the concept.

Each machine, or stack, symbolizes an operator network and each application symbolizes a node. Each stack is configured to be able to address and route messages to all of the four applications (hereby referred to as nodes). That means that any node, independently of underlying stack, can send a message directly to any other node.

The nodes are divided into two types:

- *End nodes*: These nodes symbolize any arbitrary node within a PLMN that wishes to communicate with a node in a different PLMN or operator network.

- *Security Gateways* (SEGs): The SEGs are the border-nodes where all inbound or outbound messages are routed in and out of the simulated operator networks. These nodes will contain protection mechanisms for processing the routed messages. All messages

requiring protection will traverse these nodes. No additional information will be added to or removed from the messages.

The nodes can also be classified as active or passive nodes. An active node will be under direct command of the user and will begin and terminate a dialogue. The passive node will await messages incoming from the stack. The incoming messages will be processed and forwarded, after which the passive node will return to a waiting state.

4.4.2 Prototype software design

The nodes will have different functionalities based on the type of node and its passivity. The nodes will be described in the order in which the messages arrive. Figure 4.3 illustrates an abstraction of the nodes without the underlying stack.

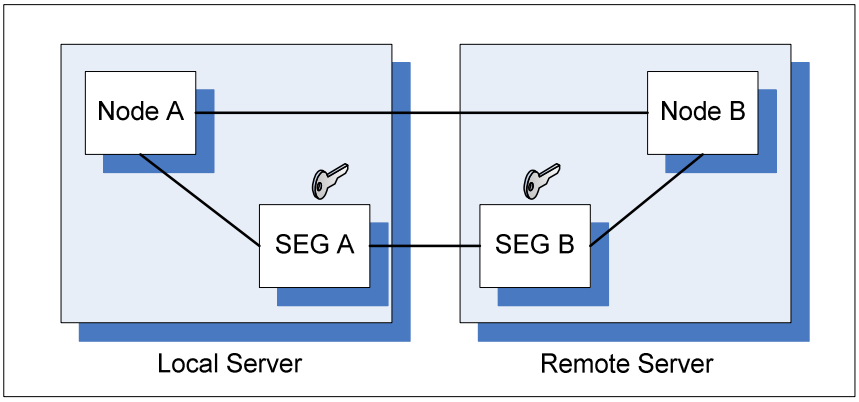


Figure 4.3: An abstraction of the prototype nodal structure and routing paths.

Node A

The first node is the active node in which the user has control over the message sending functions. All messages will be sent in individual dialogues and will start from this node. Node A will also be the terminating node, as the messages return after traversing all nodes. The traversal of the messages will be described in Section 5.2.

There are three variable parameters, which the user can set when executing the prototype:

- *Size of payload*: This parameter will set the size of the payload that is to be protected. This will set a variable load on the protection mechanisms. The maximum size of this parameter is limited by the TCAP module implementation.

- *Number of dialogues*: This is the number of dialogues that is to be sent in parallel for each test. Its purpose is to control the load on the network and the protection mechanisms. The maximum value of this parameter is limited by the SS7 stack configuration.

- *Protection mode*: This parameter signals the protection mode to the SEGs. The SEGs examines this parameter to decide on with protection mode to apply to the received message. The prototype implementation defines the values that are valid for this parameter.

The messages with payload will be sent as an Invoke request, carried by a Begin request to start a new dialogue. There are two parameters, which will store arbitrary data in the Invoke request: the *operation code* and the *operation parameter*. The payload will be stored in the *operation code* parameter and the protection mode parameter will be stored in the *operation parameter*.

When sending a message, the message will not be addressed to the final destination due to some problems with routing the messages through the SEG applications. The problem is described in Section 6.1. Instead, the message will simply be addressed directly to the neighbouring node. This solution requires that a new dialogue will be established between each neighbouring nodes. The three dialogue ids required for sending a single message would all have related dialogue ids.

In addition to the requirements in Section 4.3, another function was added: sending the message directly to the remote end node without passing the SEGs. This was implemented to examine if the actual routing through the SEGs added any significant delay to the messages. This route will only require one dialogue id per message.

Node A also has a timing functionality, measuring in milliseconds. The node starts by starting the timer and sending all the messages. Then it awaits the return of the messages from the remote end. At the arrival of the last message, the timer is stopped and the control is returned to the user. The time measured is called the *Round-Trip Time* (RTT).

SEG A and SEG B

The SEG A is the security gateway of the local simulated network. This node is passive and waits for inbound and outbound messages to process. This node has cryptographic capabilities for applying and removing protection of the messages. The capabilities will be added through the use of a cryptographic library known as “Botan” [32].

At the initialization of the node, a secret key and initiation vector (IV) is generated and stored. The generation of the key and IV is based on a passphrase known to both the SEGs. This key and IV will be used to apply or remove protection of all the passing messages that requires protection.

At the arrival of a message, the operation parameter is checked to verify the protection mode. By examining the value of the dialogue id or the originating node address, the address of the next node destination can be determined. Before forwarding the message, the payload must be processed according to the set protection mode.

- Protection mode 0: The operation code and parameter is simply forwarded to the next destination.

- Protection mode 1: The message payload will be processed with integrity mechanisms. But instead of performing the integrity calculation to get a 32-bit MAC-M block (as described in the requirements in Section 4.3), the payload is encrypted using the AES algorithm in CBC mode. The resulting ciphertext is then forwarded to the next destination.

- Protection mode 2: First the message will be encrypted with AES in counter mode, using the generated key. Then the resulting ciphertext will be processed with the same security mechanisms as in Protection

mode 1 before forwarding the message. This will result in a double encryption with two different modes of encryption.

The difference in using a MAC-M compared to simply sending the ciphertext is that the message is sent encrypted and a MAC-M is not appended to the message. Also, the time it takes to compare the MAC-M and the calculated 32-bit block at the receiving end is removed. Although the alternative solution weakens the integrity protection, the time difference caused by this alternative solution is assumed to be small enough to be disregarded.

When forwarding the message, a new message is prepared with the determined destination address. The ciphertext or the plaintext is inserted together with the operation parameters into the new message. Also, the dialogue id is converted to a relative value based on the direction of the message. When ready, the message is sent with the same primitive type as it was received (Invoke request or Result request).

When receiving a protected message the encryption process is reversed. First, the operation parameter is checked and then the message is process or decrypt according to the protection mode before forwarding the plaintext.

SEG B is almost identical to SEG A. The only difference is the address of the node in the network. Other than that, the mechanisms work in the exact same way.

Node B

The last remote node is also passive. The only function of this node is to receive incoming Begin indications followed by Invoke indications. The node answers by sending back Result requests for the Invoke indications, and End requests to end the dialogues. The Result requests will contain the same operation code and operation parameters as received in the Invoke indication. These requests are routed back along the same path to Node A for ending the dialogues and stopping the timer.

This design offers variations in form of:

- Two different routing paths.
- Three modes of protection.

- Variable load on routing and protection mechanisms.

By using this design, a difference in performance can be measured for the different protection modes, parameters, and routing paths.

4.5 Functionality testing requirements

To confirm that the functional requirements described in Section 4.3 have been met, a number of tests need to be designed to test that the prototype actually upholds these requirements. A description of these tests is stated here:

- The first part to be tested is the basic routing through the four nodes. A simple way to test this requirement is by printing out a notification as the messages arrive at each of the nodes. This printout should also show that all the messages are routed without losing any messages on the way.

- The different modes of protection need to be tested individually.

    - **Mode 0:** The basic routing will test this protection mode adequately.

    - **Mode 1:** The authentication and integrity mechanisms must be tested. This test will be carried out in two steps. The first step is to perform the integrity calculations in the SEG of the originating network and attaching the 128-bit block to the message. The second step is done in the SEG of the receiving network, where the same integrity calculations are performed on the received message and the result is compared with the attached 128-bit block.

    - **Mode 2:** The highest level of protection will be tested by performing printouts of the message in the SEGs and end nodes. The message should be printed out before and after the encryption/decryption to verify the protection mechanisms.

- Lastly, the time measurement mechanism needs to be tested. This is done by timing the routing of a single message through the serial-connected nodes. The test on time measurement must be performed on all three modes of protection without causing any major variations in the measured time within the same protection mode.

By doing the above-mentioned tests on the requirements, the prototype should be able to do a performance measurement of routing messages in all three different protection modes. The testing results of the prototype will be described in Section 4.6. The performance measurements and results will be described in Chapter 5.

4.6 Prototype testing results

The prototype was tested in accordance to the testing requirements described in Section 4.5**.** There were three main features that were tested: the basic routing of the messages, the different protection modes, and the timer functionality.

Regarding the basic routing, the prototype routes a smaller amount of unmodified plaintext messages without problems. The content of the message is printed out at each node, confirming that the whole message was routed through the node without anything added or subtracted from the contents.

Testing the protection modes was not quite as easy as it would seem. Protection mode 0 was easy to test, as it was practically the same testing routine as testing the basic routing. Protection mode 1 could not be tested as described in Section 4.5, as the prototype design did not achieve the functional requirement stated for protection mode 1 (see Section 4.4.2 when describing the SEGs). Instead of performing the calculation and comparison of the MAC, the length of the message was printed out before and after the encryption. Also, the plaintext message was printed out before the encryption and after the decryption, but the ciphertext was not. The reason for this way of printing was that printing the ciphertext caused the program and the development environment to fail. The difference in the length of the plaintext and the ciphertext showed that at least the padding mechanism of the encryption was performed on the message, which in turn indicated that the encryption mechanism was performed. Different steps in each protection mode were also printed out in each node. Protection mode 2 was tested in the same way as mode 1. The parameter used for indicating the protection mode was also printed out in each node to verify that the protection mechanism chosen in each node was based on the correct value.

To test the timer functionality, the message was simply routed with the three different protection modes and the two different routes (see Section 4.4.2) while running the timer.

4.6.1 Prototype limitations

While testing the functionality and stability, we came across a number of limitations on the prototype.

The message size in the “operation code”-parameter (see Section 4.4.2) is limited by the TCAP module implementation. The parameter was limited to 255 bytes in the configuration, but adding the padding from the encryption mechanisms and message tags (required by the TCAP module), the message size sent from the initiating node was limited to 236 bytes. This limited the operation code content to be set between zero and 236.

The SS7 stack configuration limited the number of parallel dialogues sent between the stacks. The limit was set to 1000 dialogues. When testing large amounts of parallel dialogues (approximately 300 or more), packets would be lost during routing. The packet loss would in turn cause the timer to fail. Closer examination of the signaling traffic between the stacks revealed that the load on the network caused the receiving stack to be congested. This problem is assumed to arise due to inferior servers, as the hardware used for the prototype was not dimensioned for routing large amounts of data traffic. Because of the impending risk of congestion, a delay was deliberately placed in the sending node. The duration of the delay was set to at least one millisecond between each message sent from the first node. This delay would accumulate when sending a large number of messages, for example when sending 1000 messages, a delay of at least 1000 milliseconds would be introduced into the timer result. This deliberate delay must be taken into consideration when viewing the results of the performance tests. Despite the insertion of the delay, the network would be congested when sending approximately 600 messages. Considering this upper limit, we set the number of dialogues to be between one and 500 to avoid congesting the network.

With these limitations in mind, the prototype was ready for the performance tests described in Chapter 5.

4.7 Summary

In this chapter, the creation process and the design of the prototype was presented. The design method was improvised and done in three steps. The scope of the prototype was defined and limited to the most basic protection mechanisms required to protect messages. The prototype functionality requirements were listed and the protection mechanisms were defined. With the requirements set, the hardware and software architecture of the prototype implementation was presented and each node was described in detail. A number of functionality testing requirements were defined and the results from applying these tests on the prototype implementation were presented. Also, a few limitations on the prototype implementation were discovered after performing the tests.

5 Performance evaluation study

This chapter describes the performance testing setup and procedures. Also, the performance results are presented, interpreted and discussed. The chapter begins with a presentation of the test environment configuration and parameters in Section 5.1. Section 5.2 describes the performance testing procedure when testing the parameter combinations. The results of the performance tests are presented and interpreted in Section 5.3 and finally the results are discussed and evaluated in Section 5.4.

5.1 Performance testing parameters

First we describe the testing environment configuration and the available variable parameters.

The stack software was provided by TietoEnator and is briefly described in Section 4.4.1. The hardware consisted of two UNIX SPARCengine servers with the following configuration.

```
Operative System:       SunOS 5.8
Processor:              UltraSPARC-IIi-cEngine
Primary memory:         128 Megabyte
```

The processor of the first server was specified at 360 MHz and the second server at 333 MHz. Aside from the clock frequency, the servers were almost identical. The two servers were directly interconnected with a 100 Megabit Ethernet connection to avoid burdening the TietoEnator internal network and to eliminate any external interference.

Before the testing part could be carried out, the applications needed to be started and connected to the two stacks. At first, two applications, node A and SEG A would be started and connected to the first stack. Then two other applications, node B and SEG B, would be started and connected to the second stack.

The program contains three variable parameters that can be set by the user:

- *Message length parameter*: This parameter could be set to a value between zero and 236. This given interval is due to the configuration in the stacks described in Section 4.6.1. It was decided to keep this parameter constant to 200 bytes in all the test cases and vary the other two variables instead. This decision was made to limit the number of variables when testing the performance. The value was chosen because it was assumed to be a suitable value that was not too small or too large.

- *Number of parallel dialogues*: The number of parallel dialogues was limited to a value between one and 500 dialogues due to the risk of congestion (see Section 4.6.1). The number of dialogues per test case was set to 1, 10, 50, 100, 200, 300, 400 and 500. The four protection modes were tested individually with these varying numbers of dialogues.

- *Protection mode*: The protection mode could be set to a value between zero and 3. The values 0, 1, and 2 would make the prototype route the messages through the SEGs and notify the SEGs to apply the selected protection mode mechanisms on the messages. The value 3 would instruct the prototype to set the routing path to bypass the SEGs and route the messages directly to the end node on the remote stack without applying protection. The routing paths are illustrated in Figure 5.1 below. All the protection modes were tested with all the different numbers of dialogues mentioned above.

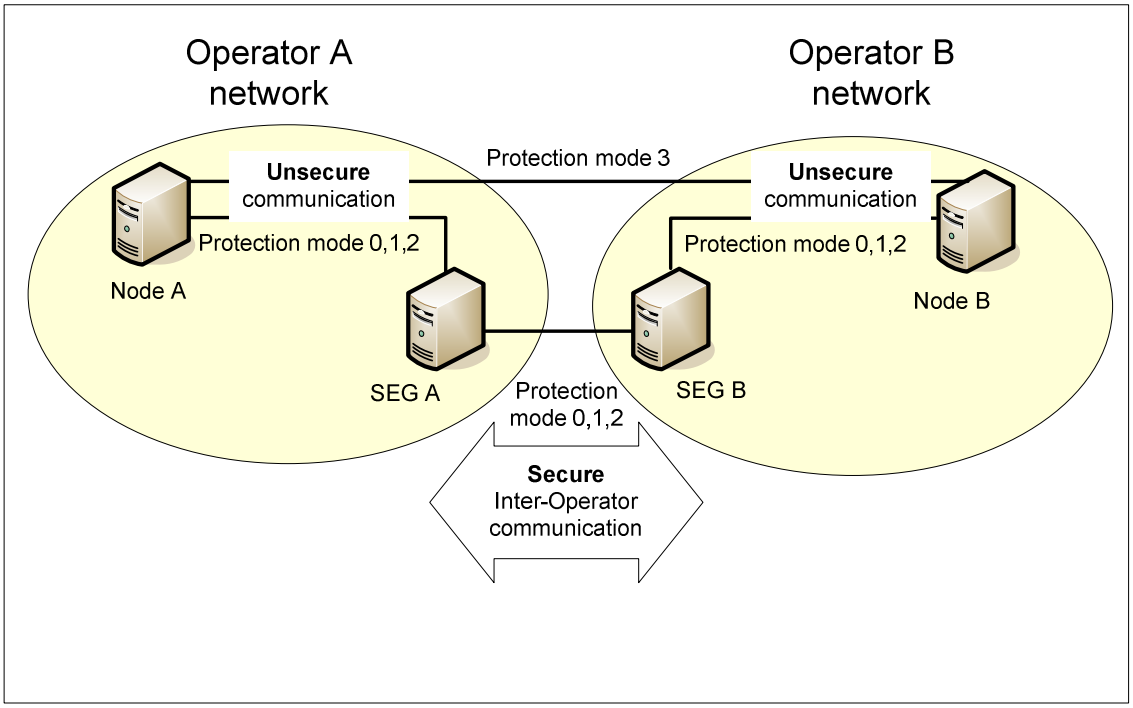


Figure 5.1: Routing paths for the different protection modes in the prototype.

5.2 Performance testing procedure description

The test was performed in the following steps:

1. Node A and SEG A, see Figure 5.1 above, was connected to stack A.
   Node B and SEG B was connected to stack B.

2. The addresses and SSN were automatically set for all four nodes according to the implementation.

3. The applications of the four nodes were automatically bound to TCAP and the simulated network was then ready for performance testing.

4. The test was prepared by setting the vectors: length of the message, the number of parallel dialogues, and the protection mode.

5. After these settings were made, a timer was automatically started and the messages were sent. The sending procedure is divided into six different steps that are described below (see Figure 5.2 below).

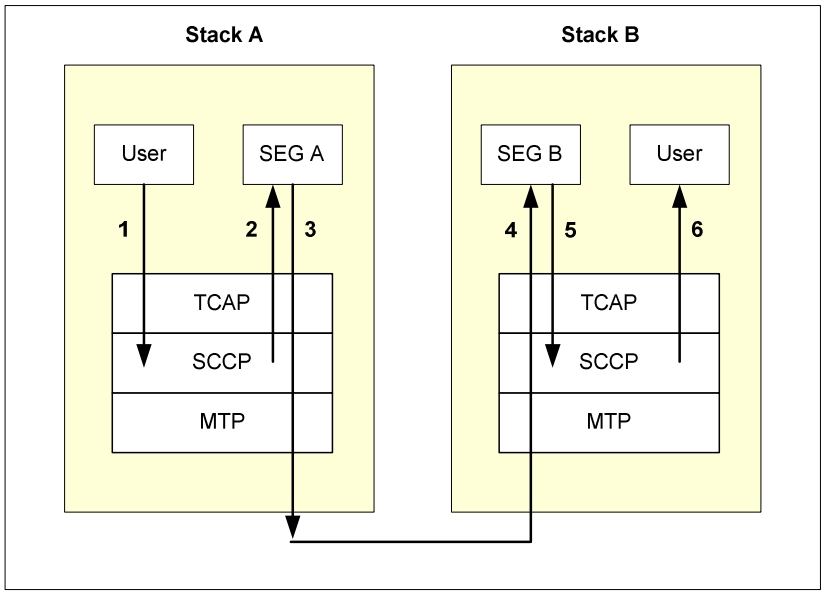


Figure 5.2: The one-way sending procedure in the stack.

1. Messages are sent from node A to SEG A.
2. Each message is treated depending on the chosen protection mode. If, for example, protection mode 2 with full encryption is used, the messages are individually encrypted in SEG A.
3. The messages are passed on to their destination, which is SEG B.
4. The messages reach SEG B and this node checks the protection mode. If, for example, protection mode 2 with full encryption is used, SEG B's task is to decrypt each message.
5. The messages are passed on to the user in Stack B, who is the receiver.
6. The user in Stack B receives the messages.

6. When the user in Stack B received a message, it was returned and the described sending procedure was repeated in the reversed order.

7. When the last message had reached the user in Stack A, the timer was stopped and the sending was completed.

A sequence diagram of sending a single dialogue can be found in Appendix D. The first diagram illustrates the case when sending in protection modes 0 to 2, which is when routing through the SEGs. The second diagram illustrates the case when addressing the remote end node directly, without traversing the SEGs. When sending more than one dialogue, the sequence of sending an Invoke_Req() followed by a Begin_Req() was repeated for the given number of dialogues. This sequence caused one invoke request (containing the payload) to be sent for each begin request, which begins a new dialogue.

5.3 Testing results

In this section, we will present and describe the test results. First the table with the average RTT of each parameter combination is described. Then the graphs based on the table are presented and described in a comparative manner.

Each test case, or parameter combination, was tested five times to get an average value of the tests. This iteration is due to seemingly random variations in the measured round-trip time (RTT). Table 5.1 shows the average values of test cases. The complete table of results is presented in Appendix B.

Number of dialogues	**RTT for each protection mode (ms)**			
	0	**1**	**2**	**3**
1	1186,8	1189,0	1187,8	384,2
10	1331,6	1176,4	1317,8	486,8
50	1173,0	1396,6	1322,0	888,0
100	1957,0	1769,4	1740,4	1385,6
200	2829,4	2860,6	2643,6	2326,8
300	3749,2	3811,6	3891,8	3406,2
400	4792,6	4850,2	4906,8	4357,6
500	5698,6	5787,8	5957,8	5447,8

Table 5.1: The mean round-trip-time for each parameter combination.

Considering the set number of iterations for each parameter combination, it is of great importance to bear in mind that it is difficult to draw any definitive conclusions based on these test results. The results and drawn conclusions in this thesis will be regarding the tested prototype, with the specific hardware and software configuration, and may indicate the behaviour of the future TCAPsec implementation.

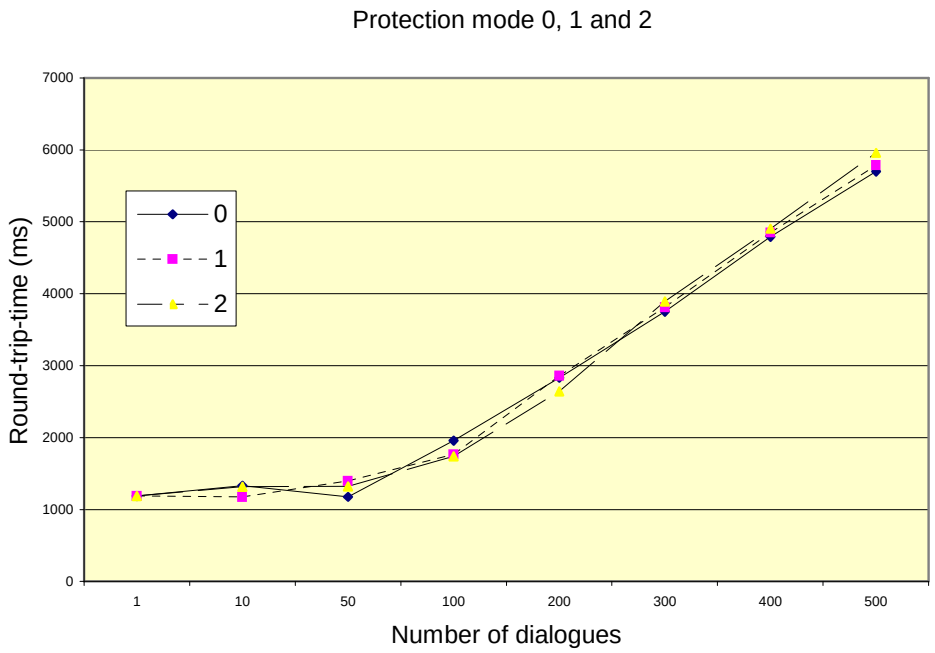


Figure 5.3: Comparison of the performance between protection mode 0, 1 and 2.

In the first graph in Figure 5.3 above we can see a comparison between the protection modes 0, 1 and 2. Common for these three protection modes is that they all route the messages through the SEGs. The graph over protection mode 0 (that is without any security mechanisms) indicates a seemingly insignificant difference when between one to 50 dialogues are sent in parallel. The curve suggests that the RTT is constant when sending 50 parallel dialogues or less. The constant RTT could be caused by an initial delay in the routing mechanisms that is greater than the actual RTT. Beyond 50 dialogues, the increase in the RTT seems to be linear relative to the number of active dialogues. This linear increase is probably due to the increase in network workload caused by the increased number of dialogues.

The graph over protection mode 1, when authentication and integrity is used, shows a similar inclination as the graph over mode 0. Despite calculations in both the sending SEG and the receiving SEG, no distinctive differences can be shown by our measurement of protection mode 0 and 1. This similarity could indicate that the protection mechanisms of protection mode 1 do not add any significant delays to the RTT.

The graph over protection mode 2 (that is the highest level of protection) indicates a slightly bigger difference in time, compared to the graph over protection mode 0, but the difference is still too small to be considered significant. An interesting observation is that the curve for protection mode 2 seems to have a slightly steeper incline at the rightmost end than the curve for protection mode 0 and 1, but this could be a mere coincidence.

Other than the minor differences in the right-hand side of the graph, protection mode 1 and 2 would seem to have very similar RTTs.

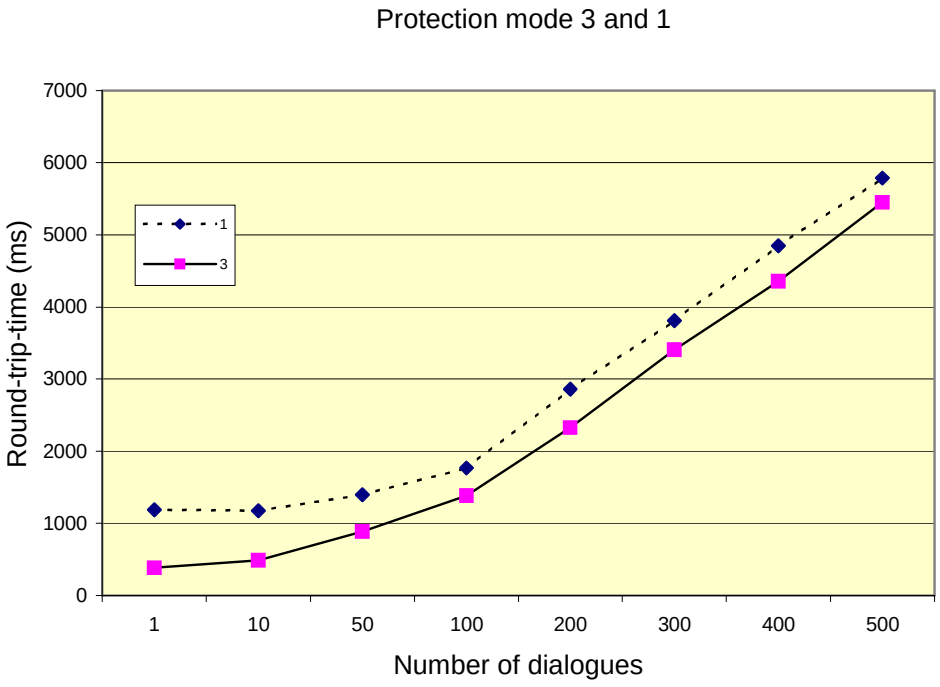


Figure 5.4: Comparison of the performance between protection mode 3 and 1.

Figure 5.4 shows the difference between protection mode 3, when routing unprotected messages directly to the remote node, and protection mode 1, when routing through the SEGs and applying integrity and authentication security to the message. Protection mode 3 in Figure 5.4 indicates a difference in time compared to using protection mode 1 in the same figure. The round-trip-time is lower when the messages are routed directly between sender and receiver. The graph indicates, just like in the previous graph, a linear increase in the RTT relative to the number of dialogues. This measurements shows that protection mode 3 routes packets faster than protection mode 1. The difference could be due to the delay caused by the routing mechanisms when routing through the SEGs. Although the time difference is palpable, it would seem that the difference is also relatively constant. The case of routing one to ten messages indicates that the extra routing links through the SEGs causes a delay difference that is greater than the time taken to route the few messages directly. This is not necessarily important as the probability of the traffic load getting this low is assumed to be relatively small.

5.4 Discussion of the results

By observing the first one to 50 parallel dialogue tests, we assume that there is an initial delay caused by the basic routing mechanisms in the stacks. This delay seems to be greater than the time taken to route less than 50 messages, which in turn causes the RTT to be the same when routing one message and routing 50 messages.

Two types of delays can be identified by the results: the delay caused by the protection mechanisms and the delay caused by the routing mechanisms. The difference in time between the protection modes 0 to 2 seem too small to be considered of greater importance, at least in the small amounts of dialogues tested. Instead, the delay caused by the alternative routing path through the SEGs seems to have a greater impact on the overall RTT than the different protection mechanisms in protection modes 0 to 2. See Figure 5.5 below.

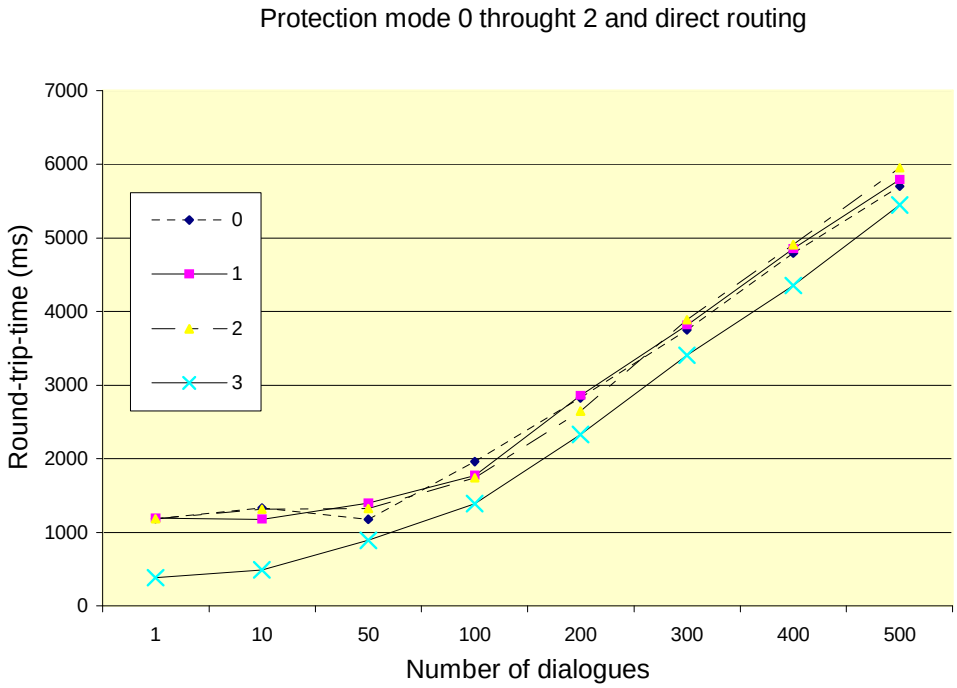


Figure 5.5: Comparison of the performance of all protection modes.

The similarities in the RTT of protection mode 0 to 2 are probably due to the efficiency of the encryption algorithm implementations. The encryption and decryption implementations where designed to process larger amounts of data than the amounts we used in the prototype. In the case of encrypting and decrypting the small data packets that the prototype uses, the key generation process would probably take more time than the encryption/decryption process. However, the key generation and management is supposed to be located in an external entity (the KAC; see Section 2.5) and is beyond the scope of this thesis. Considering the generation and storage of the SAs, the retrieval of the SAs from the SADB and the policies from the SPD could cause a significant delay penalty, but again, investigating this is not within the scope of this thesis.

In terms of time, it would be disadvantageous to use protection mode 0, 1 and 2 compared to protection mode 3. Both protection mode 0 and 3 provide insecure routing, but mode 3 has a lower RTT than mode 0. A conclusion drawn from this comparison is that there is a significant delay when routing through the SEGs. However, the delay difference is relatively small and appears to be relatively constant in terms of milliseconds. If this delay difference would

stay constant at a greater number of dialogues, the delay difference, in percent, would become insignificantly small in comparison to the total RTT when routing vast amounts of messages. Based on this assumption, the delays caused by the protection mechanisms in the SEGs would have a low significance in comparison with the delays caused by the basic routing mechanisms. If it is the routing mechanisms that cause the greatest delay, then routing the message through several nodes in a network would make the delay caused by the protection mechanisms seem even smaller. To get at clearer picture of the behaviour of the prototype, a larger number of dialogues should be tested (see Section 7.2).

The added delay from an alternative routing path would indicate that the deployment of gateways and configurations of the routing paths in an operator's network would cause the greatest delay, as the messages must traverse the gateways instead of getting routed directly to the remote destination. It would then be most vital to dimension the network accordingly to set up enough security gateways to avoid unnecessarily large deviations in the routing path compared to direct routing.

The conclusions drawn above are based on the results when measuring the RTT. What the prototype does not measure is the additional workload put on the processors in the nodes. This measurement might be necessary in the future to properly dimension the networks for handling the protection mechanisms. See Section 7.2.

One interesting observation is the slightly steeper incline of protection mode 2 when routing more than 300 dialogues. When looking closely at the graphs in Figure 5.3, a very small but visible increase in difference of the RTT, in comparison to protection mode 0 and 1, is noticeable. If this increase is kept constant, the difference might grow larger and larger when sending greater numbers of dialogues. If the difference does increase, it would indicate that the protection mechanism could cause a very large, or even intolerable, delay in the RTT when routing vast amounts of dialogues. If this is true, the delay might be countered by deploying more gateways to lower and balance the overall load on the gateways. This protection mode should be thoroughly tested to verify or repudiate the assumption. See Section 7.2.

To summarize the discussion, we can say that two seemingly dominant delay sources have been identified: the protection mechanisms and the routing mechanisms. By comparing the delay penalties caused by these two sources, we can say the implemented protection mechanisms do not produce any major delay penalties. Instead, the routing mechanisms would seem to cause a more significant delay penalty than the protection mechanisms.

6 Problems and experience gained

This chapter contains the encountered problems and the experience gained during the thesis work. Section 6.1 presents the problems we encountered during the thesis work and Section 6.2 summarizes the gained experiences.

6.1 Problems

While writing the thesis, we encountered a number of problems. The problems were not only of technical nature, but also formal and physical. Here are some of the problems stated with a short description of the solution we used.

In Section 4.1, we describe the design method chosen to design the prototype. In retrospect, this might not have been such a good choice. In the initial phase of the design, when investigating the requirements on the prototype, the level of confusion was very high. Due to the relatively improvised and unstructured design method, we were uncertain in how to conduct the requirements investigation properly. This uncertainty caused more time to be spent on the design process than necessary. We are still uncertain if the extra time spent were more than the time it would take to study and implement the UP or XP procedures. But if we did follow the methodologies, it would definitely result in a much better and more detailed documentation of the prototype.

During the work of designing the TCAPsec prototype, a number of implementation problems occurred. Most of the discovered problems where solved with a compromising solution due to the scope and time limitations of the thesis work. Many of the problems will be mentioned again in Section 7.2 to encourage correction of the compromising solutions and implementation of proper solutions of the problems. The discovered problems and the temporary solutions are described below.

- The biggest problem is in the routing aspect. Due to the design of the TCAPsec security gateway concept, the messages are supposed to route upwards to the TCAP layer in the security gateways (SEGs). This way of routing conflicts with the design of the TCAP layer and the routing mechanisms of the SCCP layer (see Section 2.3.2). Firstly, TCAP was designed to be an end-to-end protocol, which implies that the layer is only involved in the end nodes (that is, the sending node and the receiving node). When introducing SEGs into the network, the end-to-end concept will be broken, as the messages must be routed to the TCAP layer in the intermediate SEGs. Secondly, the SCCP was designed to address the remote destination node directly. Forcing the message to be routed up to the TCAP layer in the intermediate SEGs could imply that changes has to be made in the SCCP address configuration. Since the SCCP addressing procedures are highly complex and out of the scope of this thesis, the problem was not properly solved. Instead, a compromised solution was implemented. In this solution, the nodes only have knowledge of the address of the neighbouring (end or SEG) nodes. The node could then create connections to the neighbouring nodes. With this solution, the path to the remote node was created by linking connections between the nodes into a chain. By using this solution, the addressing transparency[4] is broken and the message can only traverse a predefined path defined in the implementation. This is an inflexible solution, but it works for a small prototype such as ours. A suggested solution was to modify the routing tables in the SCCP layer to force the messages to be routed towards a SEG regardless of the set destination address, and up to the TCAP layer before leaving or entering the protected network.

- During the implementation and continuous testing of the prototype, an error was discovered in the SS7 protocol stacks provided by TietoEnator. The original plan was to use a parameter in the begin request primitive called *user information* as a payload carrier. The user information parameter is used to exchange information between TCAP users and could contain an arbitrary data payload. Together with the payload, the begin request would single-handedly start a dialogue with the neighbouring node while carrying the message payload and no other primitive would have had to be sent from the node. However the parameter did not work as planned. By reading the logs generated

[4] Addressing transparency is upheld when the nodes can address the messages to the remote end node without having to be aware of the message traversing the intermediate SEGs.

by the SS7 stack manager, it was discovered that the data in the parameter still existed when sending the message from the application to the TCAP module in the SS7 stack. When forwarding the message from the TCAP module to the SCCP module, the user information data was not included in the primitive. This was identified and reported as a bug in the TCAP module. The alternative solution used was to send an invoke request containing the payload along with the begin request (see Section 4.4.2). It was later discovered that this solution might not be a very realistic one according to the TCAPsec specification. See Section 7.2 for more information on this issue.

- When executing the performance tests, the stacks acted very unpredictably. At first, when the prototype was halfway implemented, we believed that the behaviour was normal. When testing the complete prototype, the unpredictable behaviour would prove to cause unreasonably large delays. The delays would be so large that they were longer than the actual RTT of the messages. This problem was caused by inefficient configurations in the stacks. The stack configurations were beyond the scope of this thesis, and the scope of our knowledge. Therefore, our supervisors at TietoEnator had to provide us with the solution. The supervisors removed the major delays in the configurations and optimized them as much as possible. The stacks were still a bit unstable at the time of the performance tests, but the variations were considerably smaller than before the optimization. By performing enough tests and calculating the result statistically, the results could provide a fairly accurate picture of the prototype's performance. See Section 7.2.
- The testing hardware (two UNIX servers) provided by TietoEnator does not appear to be powerful enough to perform larger tests. When testing the finished prototype with the optimized configurations (described in the previous point), it was discovered that the traffic line between the two servers would get congested at higher loads than 300 dialogues. The first solution to this problem was to introduce a delay between the sending of the dialogues. This delay was set to one millisecond. Even though the delay was introduced, the line would still get congested at 600 dialogues. It was decided to limit the tests to a maximum of 500 simultaneous dialogues. It should be mentioned that the stack configurations were specified to a maximum of 1000 simultaneous dialogues. Also, depending on the location of a node in a real network, the load could get up to over 60000 dialogues.

- The TCAPsec technical specification does not specify exactly where in the SS7 stack the solution should be implemented. Either the solution could be implemented as a TCAP application above the TCAP layer, interfacing the TCAP API, or it could be integrated into the TCAP layer itself. On one hand, using the TCAP API would signify a considerable limitation to the TCAPsec possibilities. On the other hand, the TCAP implementation is enormous and its documentation is over 1000 pages long. After discussing the subject with our supervisors at TietoEnator, it was decided that our prototype would be designed as a TCAP application, utilizing the TCAP API. Just like the payload problem mentioned above, it was later discovered that this might not be a realistic solution. See Section 7.2 for more details.

6.2 Experience gained

The extent of this Master's thesis is larger than anything else we have done before. Since we did not have any knowledge of SS7, it took a long time to get an overview of the task. We had to perform an extensive research part. This was a great challenge since it was a large amount of literature to read and it was not easy to sort out the essential information for our work. From the study and programming work, we have learned that it is very important with a precise and detailed documentation of the API and source code.

From the beginning of our work, we made an extensive plan for the solution of the prototype design. The lack of experience made it very difficult to know where to set the limits of our work. However, as the work progressed, we decided on a suitable scope and learned to limit this scope as we got closer and closer to the deadline.

One of the most interesting aspects of this thesis work was to experience the differences between the studying situation at the university and the real-life working situation. One of the most palpable differences between the environments and situations is that a simple or exact solution is not necessarily available in the working situation. Instead, the solutions to the problems sometimes have to be improvised. This can be compared with the studying situation where, most of the times, there is an easily accessible key or a teacher with the correct answers.

We have also had the opportunity to experience the fast paced changes that can occur in an active project. The workplan for this thesis has changed several times during the progress of the work. This caused a large amount of work to become unnecessary, as the result from the work was no longer within the scope of the new workplan. This has not only been a bad experience, but also a very educational one as this is more realistic in a real-life working situation than the strict workplan specifications used in the universities. The fast paced changes show the importance of an open dialogue, good communications, and clear goals.

7 Summary

In this chapter we present a summary of the drawn conclusions of the whole thesis work and a number of recommendations for future work. In Section 7.1 we present the conclusions and in Section 7.2 the future work recommendations are described.

7.1 Conclusions

Computer security is a hot topic these days and sensitive information is constantly sent in networks around the world. This has attracted an increasing number of fraudulent actors. However, due to the growing problem with unauthorized access to sensitive information, it is likely that security will be implemented in networks to make communication and signaling more reliable in the future.

We have described a security concept called TCAPsec that introduces protection mechanisms to the SS7 signaling network. We have also shown that it is possible to implement a prototype of this concept. We have made a prototype, which provides security for messages in the signaling network. The prototype incorporates the use of security gateways (SEGs), which are limited to the most basic protection mechanisms specified in the TCAPsec specification. The basic mechanisms consist of integrity calculations and encryption/decryption. The prototype SEG is designed as an application to the TCAP layer and utilizes the TCAP API. This design implies some limitations to the SEG's range of protection, see Section 7.2 for details. Messages can be routed between different nodes with protection, but there are some limitations to the testing environment. There are, for example, restrictions affecting the maximum number of parallel dialogues. When a large number of dialogues are sent in parallel, congestion occurs in the network and packet loss is a fact. Another limitation in our prototype is the lack of SA negotiation and distribution mechanisms. This omission is due to the overwhelming workload required to realize them. This problem is further discussed in Section 7.2.

With a complete implementation of the prototype design, the performance penalties could be studied. We have made an evaluation study of the penalties caused by the encryption/decryption mechanisms when introducing TCAPsec. When studying the costs of routing each message via SEGs, with or without protection, the results indicated no significant difference when sending protected or unprotected messages. Instead, the results indicate that there is a difference between routing messages via a SEG or directly. When routing directly between two nodes, the RTT is shorter than if the messages were routed via a SEG. Another result, drawn by studying the graphs over different protection modes, is that when the number of dialogues increases, the RTT increases linearly.

The summarized results of the performed tests indicate that the routing process in the network causes a more significant delay in the RTT, compared to the delay caused by the protection mechanisms. That is, the delay-induced cost of the protection modes is smaller than the cost of the alternative routing path. The results were different than expected. We had imagined that the protection mechanisms would be responsible for the greater delay in the RTT when using TCAPsec in the network. In retrospect, the result might have been different if the prototype was tested with a greater amount of parameter combinations and iterations.

Working on this thesis has lead to an identification of a number of items that can be closer examined, see the recommendations in Section 7.2.

The main purpose of this thesis, examine the feasibility of implementing TCAPsec and measure the performance costs of using TCAPsec, is fulfilled. It should be mentioned that the results are only valid when using the specified hardware and software. In a larger network and using another hardware setup, the results could be different. See Section 7.2.

7.2 Recommendations for future work

Considering the limitations that were put on the prototype, there are still many aspects to expand and test. Below are a few points that could be looked into when developing the prototype.

- A suggestion would be to correct the protection mechanisms for protection mode 1. That is, to use MAC-M instead of simply encrypting the message to make this functionality more accurate according to the TCAPsec technical specification in [17] and

[19]. There is a slight risk that the, rather crude, solution of protection mode 1 in the current design could cause an unexpected and most significant difference in the delay compared to the MAC-M solution. Also, the use of the time variant parameter (TVP) should be implemented into all protection modes (see Section 3.2.2). The load added to the CPU and network by the TVP will probably not cause any considerable delays in the performance of the SEGs.

- Another point of interest would be to change the testing environment to a larger and more stable network with greater capacity. This change would enable the prototype to be tested with a greater workload (with more dialogues and messages) and a larger number of end nodes. Another aspect of this is to see if multiple end nodes would have any unforeseen side effect on the performance.

- It would be interesting to see more accurate results from the prototype by testing the prototype exhaustively to get a greater number of test results. The test results should then be calculated statistically to acquire a more accurate picture of how the prototype actually performs. If the testing environment were to be expanded, as described above, then a greater number of dialogues should also be tested. An interesting detail in these results would be to see if the RTT of protection mode 2 has a greater increase in time compared to mode 0 and 3. See Section 5.4 for a description of this observation.

- When changing or extending the testing network, the problem with routing must be addressed. The addressing should be changed to global title (GT) routing instead of the direct point code (PC) routing used on the current prototype. See Section 6.1 for a more detailed description of the problem and a suggested solution.

- When designing the TCAPsec prototype, some elements that could potentially cause significant costs were left out. The Security Policy Database (SPD) and Security Association Database (SADB) are examples of such elements. The data query and retrieval time of the databases could cause major delays as they are required to be performed frequently. Such elements should therefore be implemented and tested to investigate the cost significance.

- When testing the prototype, the only thing measured in this thesis was the round-trip time (RTT) with a defined encryption algorithm. Aside from the RTT, the workload in

terms of CPU load and memory usage on the SEG servers would also be of interest. Other encryption algorithms might also be of interest to test as some algorithms are faster than others and some exerts a smaller load on the processor.

The prototype was originally intended to simulate the basic functionality of a complete TCAPsec solution. While designing the prototype with the restrictive limitations, a number of implementation aspects were uncovered that was not included into the prototype design due to the limitations. Those aspects may have to be taken into consideration when designing and implementing the complete TCAPsec solution. Below are a few aspects and details described.

- As mentioned above, the TCAP layer concept was to provide a completely transparent end-to-end service to the upper layer applications. When deploying the TCAPsec, the transparency might not be maintained, as the messages have to be routed up to at least the TCAP layer in the SEGs. This routing needs to be solved somehow. See Section 6.1 on a description of this problem.

- The prototype of TCAPsec was implemented as an external application to the TCAP layer. This solution might not be the same as the complete TCAPsec solution, because the TCAP user security specification does not specify exactly what part of the message or dialogue that is to be encrypted. Since the prototype is implemented as an application to the TCAP module, the implementation is thereby limited by the TCAP API. That is why only the payload in the invoke request is encrypted. There are three suggested solutions on how the SEG could be implemented. The first suggestion is to implement it as the prototype and somehow adapt to the API limitations. The second suggestion is to keep the SEG implementation as a TCAP application, but expand the TCAP API to give the security application greater access and possibility to protect the whole dialogue instead of just the internal parameters of the message. The third suggested solution is to integrate the TCAPsec functionality directly into the TCAP layer itself, creating a specialized security gateway version of the TCAP layer. The first suggestion would seem to be the simplest, but least effective. The second and third solution could be more complex. The complexity of expanding the TCAP API or integrating the TCAPsec functionality into the TCAP module needs to be investigated to see if it is feasible.

- A major part required to realize the TCAPsec concept is the Security Association negotiation and distribution. The key generation and secure distribution is not yet defined and must be implemented to complete the TCAPsec realization. If a centralized key administration center is to be used, the center must be designed and the generation and inter-operator distribution routines (the “Ze-interface” mentioned in Section 2.5) must be defined. This part will probably require the most work before completion of the TCAPsec realization.

References

[1] Wikipedia. http://sv.wikipedia.org/wiki/Antonio_Meucci.
Retrieved December 12, 2005.

[2] Wikipedia. http://sv.wikipedia.org/wiki/Telefon.
Retrieved December 12, 2005.

[3] A. Olsson, M. Narup, C. Helgeson, T. Eriksson, L. Lundberg, A. Lindberg, J. Carlbom, U. Vallenor, L. Bergquist, S. Johansson. *Att förstå telekommunikation.* Lund: Studentlitteratur, 1996.

[4] TietoEnator Internal SS7 course material.

[5] L. Dryburgh and J. Hewett. Signaling System No. 7 (SS7/C7): Protocol, Architecture, and Services. Indianapolis USA: Cisco Press, 2005.

[6] Intel Signaling System 7 Solutions. *Enabling Intelligent and Wireless Networks... Voice Portals and Beyond.* USA: Intel Corporation, 2001. White Paper.

[7] G. Redmill. *An Introduction to SS7.* USA, 2001. White Paper.

[8] J.F. Kurose and K.W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* USA: Addison Wesley, 2001.

[9] P. Gralla. *How wireless works.* Indianapolis USA: Que Publishing, 2002.

[10] Protocol Dictionary. http://www.protocols.com
Retrieved December 12, 2005.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. *Hypertext Transfer Protocol -- HTTP/1.1*. RFC 2616. IETF, 1999.

[12] J. Postel. *Simple Mail Transfer Protocol.* RFC 821. IETF, 1982.

[13] J. Postel, J.K. Reynolds. *File Transfer Protocol (FTP).* RFC 959. IETF, 1985.

[14] J. Postel. *Transmission Control Protocol.* RFC 793. IETF, 1981.

[15] J. Postel. *User Datagram Protocol.* RFC 768. IETF, 1981.

[16] T.Russel. *Signaling system #7, third edition*. USA: McGraw-Hill Companies, Inc, 2000.

[17] Technical Specification, release 7, version 1.0.0. *TS 33.204 - TCAP user security*. 3GPP Organization Partners, 2005.

[18] W. Stallings. *Network Security Essentials, 2nd Edition*. USA: Prentice Hall, 2002.

[19] Technical Specification, release 7, version 0.1.0. *TS 33.204 - TCAP user security*. 3GPP Organization Partners, 2005.

[20] FIPS Publication 197. *Specification for the Advanced Encryption Standard (AES).* 2001.

[21] FIPS 800-38A. *Recommendation for Block Cipher Modes of Operation.* 2003.

[22] Technical Specification. *ISO/IEC 9797 - Information technology -- Security techniques -- Message Authentication Codes (MACs) -- Part 1: Mechanisms using a block cipher.* ISO Standard, 1999.

[23] Technical Specification, release 6. *TSGS#24(04)0279 - Revitalization of MAPsec specification work.* South Korea: Technical Specification Group Services and System Aspects, 2004. Work in progress.

[24] Technical Specification, release 5, version 5.5.0. *TS 33.210 – IP network layer security*. 3GPP Organization Partners, 2003.

[25] Technical Specification, release 6, version 6.1.0. *TS 33.200 – MAP application layer security*. 3GPP Organization Partners, 2005.

[26] Technical Specification, release 4, version 4.3.0. *TS 33.200 – MAP application layer security*. 3GPP Organization Partners, 2001.

[27] Technical Specification. *S3-040581 - SMS Fraud countermeasure.* Acapulco, Mexico: 3GPP Organization Partners, 2004. Work in progress.

[28] Technical Specification. *S3-040802 - SMS Fraud countermeasure*.
St Paul's Bay, Malta: 3GPP Organization Partners, 2004. Work in progress.

[29] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality.* Gaithersburg: NIST, 2004.
Special Publication.

[30] K. Scott. *The Unified Process Explained.* USA: Addison Wesley, 2001.

[31] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change 2nd Edition.* USA: Addison Wesley, 2004.

[32] Crypto library Botan. http://botan.randombit.net
Retrieved December 16, 2005.

Appendix

A Abbreviations

AE	Application Entity
AES	Advanced Encryption Standard
API	Application Programming Interface
BSC	Base Station Controller
CBC	Cipher Block Chaining
CC	Country Code
CHA	Component Handling
CL	Connectionless
CO	Connection-oriented
CSL	Component Sub-Layer
CTR	Counter
DHA	Dialogue Handling
DPC	Destination Point Code
FTP	File Transfer Protocol
GSM	Global System for Mobile Communication
GT	Global Title
GTT	Global Title Translation
HLR	Home Location Register
HTTP	Hypertext Transfer Protocol

IETF	Internet Engineering Task Force
IMSI	International Mobile Station Identity
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISO	International Standard Organization
ISUP	ISDN User Part
ITU	International Telecommunication Union
IV	Initiation Vector
KAC	Key Administration Center
MAC	Message Authentication Code
MAP	Mobile Application Part
MAPsec	Mobile Application Part security
MS	Mobile Station
MSC	Mobile services Switching Center
MSISDN	Mobile Station Integrated Service Digital Network
MTP	Message Transfer Part
NDC	National Destination Code
NE	Network Entity
NSP	Network Service Part
OPC	Originating Point Code
OSI	Open System Interconnection
PC	Point Code
PLMN	Public Land Mobile Network
PSTN	Public Switched Telephone Network
RTT	Round-Trip Time
SA	Security Association
SADB	Security Association Database

SCCP	Signaling Connection Control Part
SCP	Service Control Point
SEA	SS7 SEG Encryption Algorithm identifier
SEG	Security Gateway
SEK	SS7 SEG Encryption Key identifier
SIA	SS7 SEG Integrity Algorithm identifier
SIGTRAN	Signaling Transport
SIK	SS7 SEG Integrity Key identifier
SM	Short Message
SMS	Short Message Service
SMSC	Short Message Service Center
SMS-IWMSC	Short Message Service Interworking MSC
SMTP	Simple Mail Transfer Protocol
SP	Signaling Point
SPC	Signaling Point Code
SPD	Security Policy Database
SPI	Security Parameter Index
SS7	Signaling System No. 7
SSN	Subsystem Number
SSP	Service Switching Point
STP	Signal Transfer Point
TCAP	Transaction Capabilities Application Part
TCAPsec	TCAP user security
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TSL	Transaction Sub-Layer
TUP	Telephony User Part

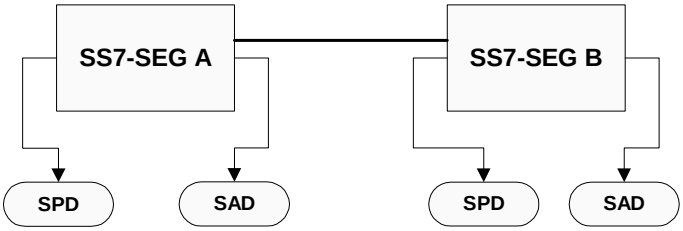
TVP	Time Variant Parameter
UDP	User Datagram Protocol
UP	Unified Process
VLR	Visitor Location Register
XP	Extreme Programming

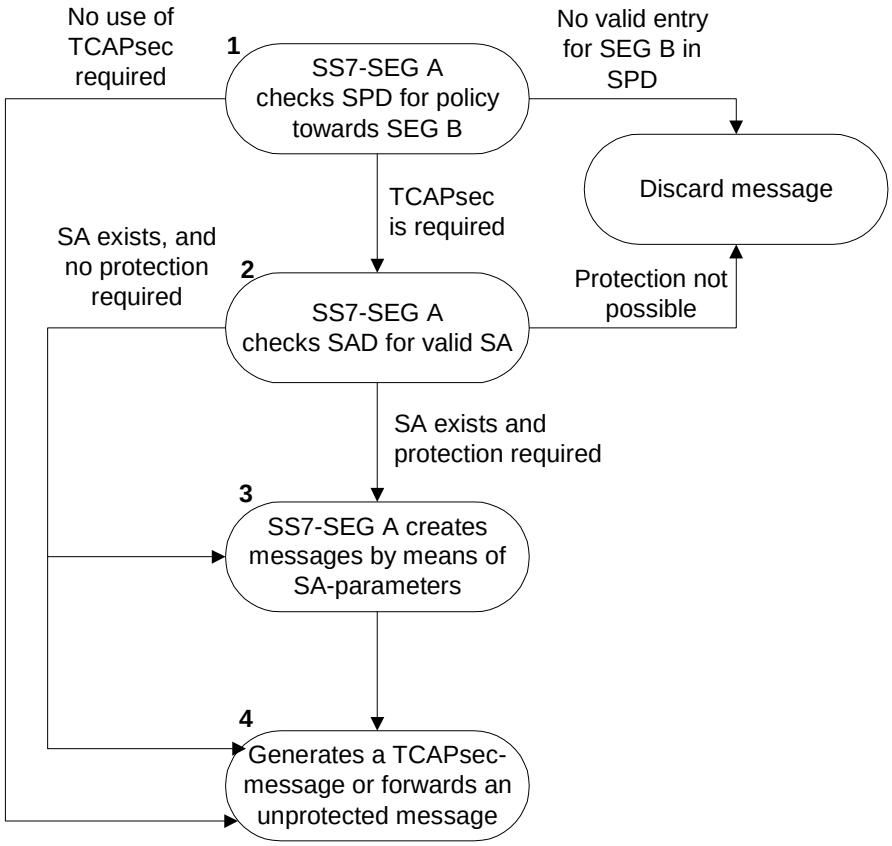
B Testing results

Number of dialogues	RTT for each protection mode (ms)			
	0	1	2	3
1	1198	1203	1188	388
	1190	1190	1188	381
	1183	1184	1190	384
	1181	1181	1188	381
	1182	1187	1185	387
	1186,8	**1189**	**1187,8**	**384,2**
10	1183	1102	1397	409
	1205	1195	1295	509
	1390	1193	1295	401
	1389	1200	1300	507
	1491	1192	1302	608
	1331,6	**1176,4**	**1317,8**	**486,8**
50	1161	1451	1162	886
	1055	1283	1396	894
	1286	1604	1438	887
	1086	1224	1350	885
	1277	1421	1264	888
	1173	**1396,6**	**1322**	**888**
100	1881	1993	1587	1386
	1974	1872	1776	1390
	1974	1588	1791	1383
	1977	1976	1775	1384
	1979	1418	1773	1385
	1957	**1769,4**	**1740,4**	**1385,6**
200	3000	2973	2613	2389
	2988	2778	2608	2382
	2590	2587	2791	2288
	2973	2980	2598	2285
	2596	2985	2608	2290
	2829,4	**2860,6**	**2643,6**	**2326,8**
300	3739	3928	4202	3421
	3847	3834	3761	3412
	3728	3694	3674	3362
	3737	3994	4055	3419
	3695	3608	3767	3417
	3749,2	**3811,6**	**3891,8**	**3406,2**
400	4622	4570	4991	4319
	4849	4880	4733	4419
	4829	4844	4978	4313
	4840	5228	5064	4420
	4823	4729	4768	4317
	4792,6	**4850,2**	**4906,8**	**4357,6**
500	5744	5874	6265	5454
	5588	5728	5924	5495
	5706	5789	5800	5488
	5696	5637	5886	5483
	5759	5911	5914	5319
	5698,6	**5787,8**	**5957,8**	**5447,8**

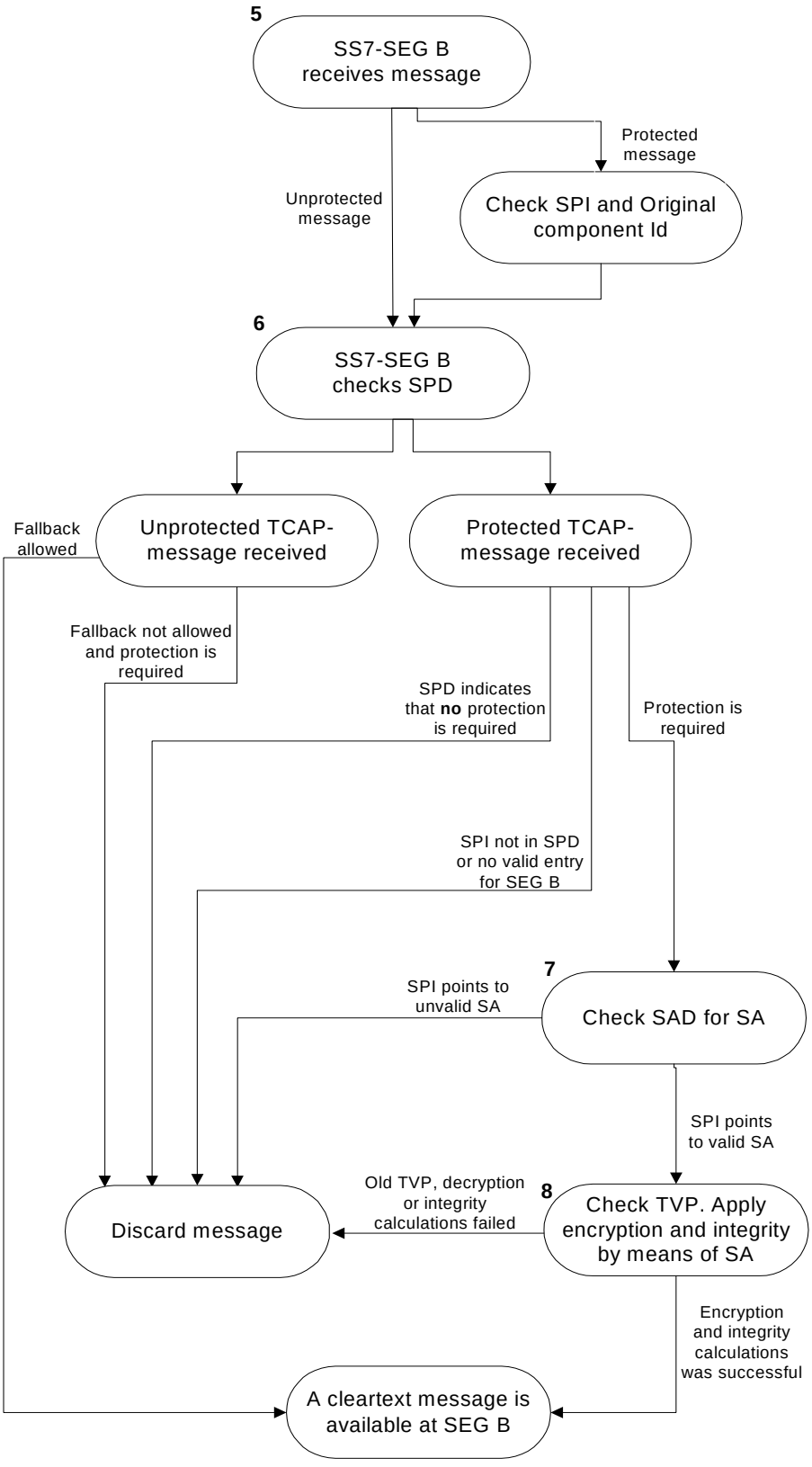
C Message flow chart

Sender



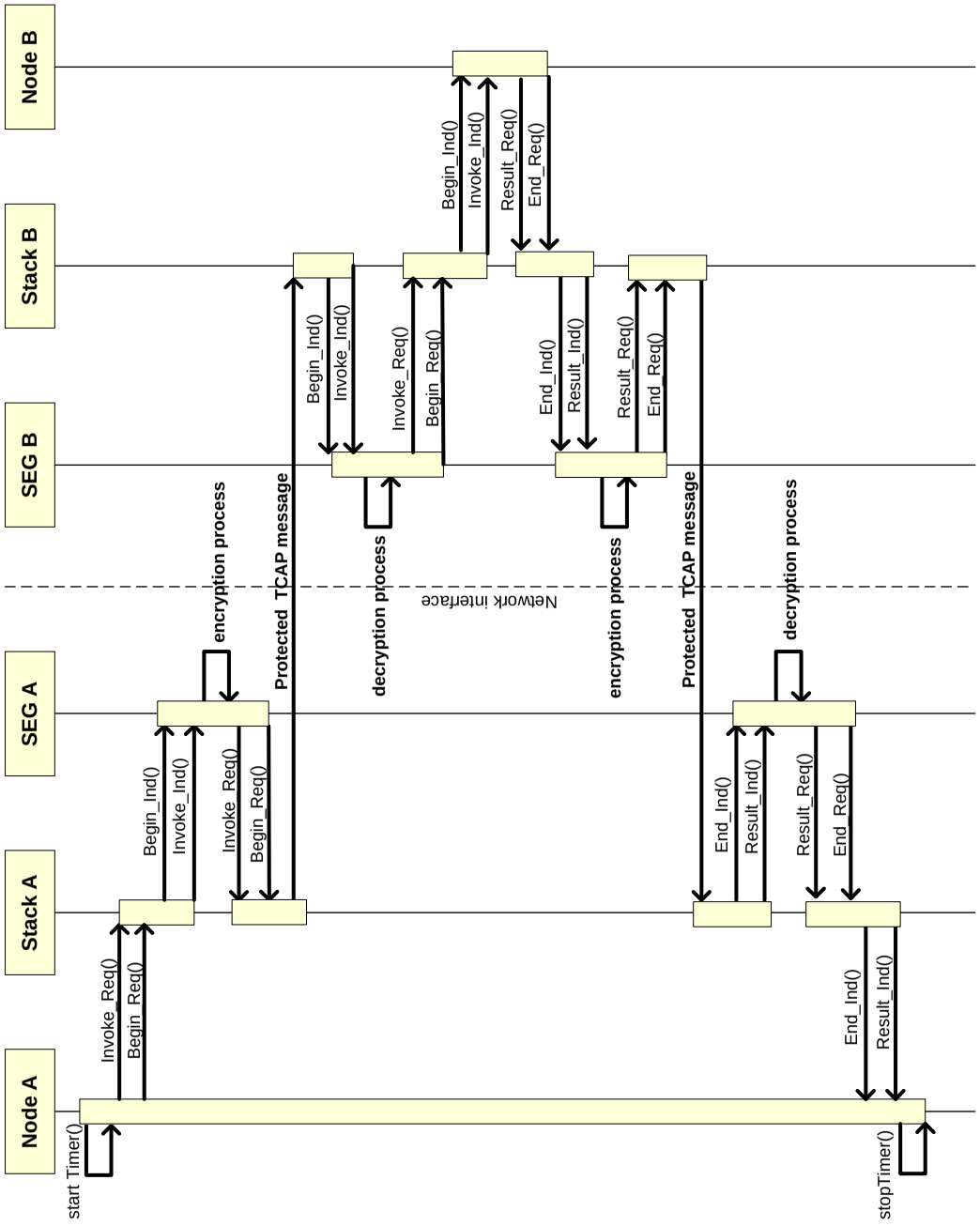


Receiver

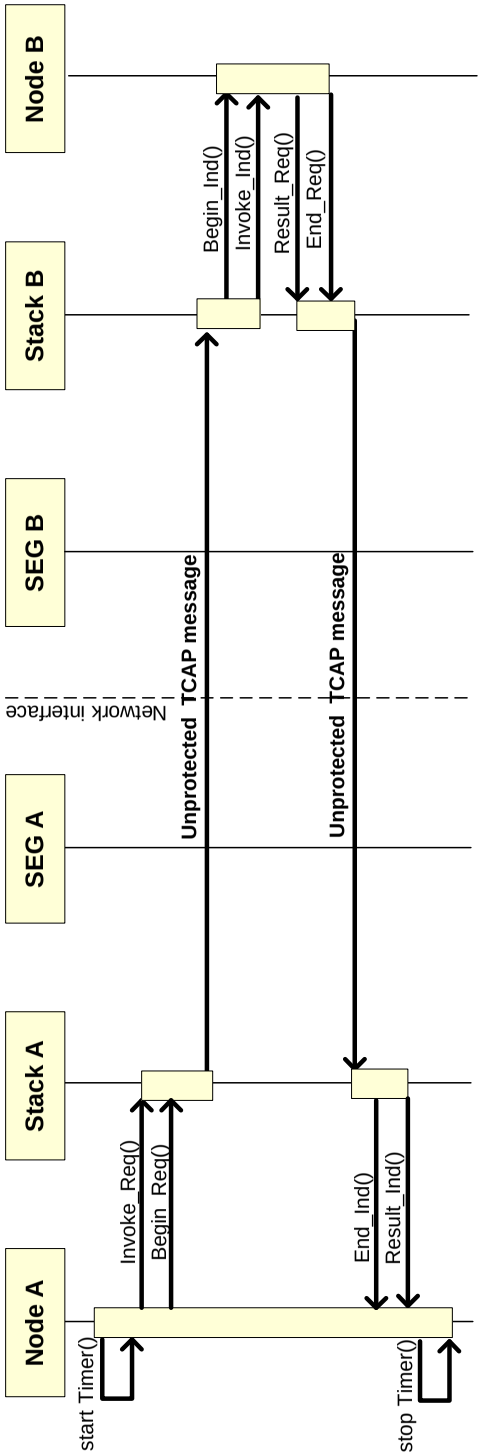

5
SS7-SEG B receives message
Protected message
Unprotected message
Check SPI and Original component Id
6
SS7-SEG B checks SPD
Fallback allowed
Unprotected TCAP-message received
Protected TCAP-message received
Fallback not allowed and protection is required
SPD indicates that **no** protection is required
Protection is required
SPI not in SPD or no valid entry for SEG B
7
SPI points to unvalid SA
Check SAD for SA
SPI points to valid SA
Old TVP, decryption or integrity calculations failed
8
Discard message
Check TVP. Apply encryption and integrity by means of SA
Encryption and integrity calculations was successful
A cleartext message is available at SEG B

D Sequence diagram

Protection mode 0, 1 and 2



Protection mode 3



E Source code

```
/*=========================================================================*/
/*
 * File: tcapSec.c
 *
 * Copyright (C) TietoEnator AB 2005 - All rights reserved.
 *
 * No part of this program may be reproduced in any form without the written
 * permission of the copyright owner.
 *
 * The contents of this program are subject to revision without notice due to
 * continued progress in methodology, design and manufacturing.
 * TietoEnator shall have no liability for any error or damage of any kind
 * resulting from the use of this program.
 *
 * Authors: Kang Chung & Mathilda Gustafsson. October 28, 2005.
 * Skeleton provided by: Daniel Rååd. October 27, 2005.
 */

/*=========================================================================*/

/*=========================================================================*/
/*                        I N C L U D E S                                  */
/*=========================================================================*/

#include <botan/botan.h>          /* Includes the AES library */
#include <sys/time.h>

#include "ss7cp.h"
#include "i97tcapapi.h"
#include "string.h"

/*=========================================================================*/
/*                        C O N S T A N T S                                */
/*=========================================================================*/

#define MSG_TRACE_ON       FALSE

#define BYPASS_SEG         3

#define PASSPHRASE         "passphrase"
#define MESSAGE            "!This is an arbitrary message of arbitrary length!"
#define INVOKE_TIMEOUT     100
#define NOOFITERATIONS     1024 //4096
#define SIZEOF_KEY_AND_IV  16
#define INTEGR_ALGOR       "AES/CBC/PKCS7"
#define CRYPTO_ALGOR       "AES/CTR-BE"

/*=========================================================================*/
/*                    G L O B A L  V A R I A B L E S                       */
/*=========================================================================*/

using namespace Botan;

/*W16 UserId = USER01_ID;*/
W16 UserId;
UCHAR_T Ssn; /* = 20;*/

UINT_T Testcase;
INT_T Running = 0;
INT_T Bound = 0;

ULONG_T noOfParallelDialogues = 0;

USHORT_T dialogueID; /* Between 1-65535 */
UCHAR_T  invokeID;

EINSS7INSTANCE_T tcapInstanceId = 0; /* Only one instance */
UCHAR_T priOrder = 0; /* Olika värden på sid 51 i TCAP API */
```

```
UCHAR_T qualityOfService = 1;
UCHAR_T destAdrLength = 4;
UCHAR_T orgAdrLength = 4;
UCHAR_T appContextLength = 0;
UCHAR_T *appContext_p = NULL;

int noOfRecvs = 0;

UCHAR_T nodeAdrLength = 4;
UCHAR_T myNodeAdr[4] = { 0x43, 0, 0, 0 };
/* Observe: the octet order of the address */
UCHAR_T node1Adr[4] = { 0x43, 0x34, 0x12, 0x14 };
UCHAR_T node2Adr[4] = { 0x43, 0x34, 0x12, 0x13 };
UCHAR_T node3Adr[4] = { 0x43, 0x47, 0x03, 0x09 };
UCHAR_T node4Adr[4] = { 0x43, 0x47, 0x03, 0x15 };

INT_T invokes = 0;
INT_T results = 0;

/* Crypto globals */

SymmetricKey* key = NULL;
InitializationVector* iv = NULL;

/*============================================================================*/
/*                 F U N C T I O N  P R O T O T Y P E S                        */
/*============================================================================*/

#if TRUE
#define debug_printf(X)
#else
#define debug_printf(X) printf X
#endif

#define printf_array(prefix, ptr, len)         \
   if (((len) > 0) && ((ptr) != NULL))         \
   {                                           \
      int idx;                                 \
      printf("%s", (prefix));                  \
      for (idx = 0; idx < len; idx++)          \
      {                                        \
         printf("%c ", ptr[idx]);              \
      }                                        \
         printf("\n");                         \
   }

const UINT_T InvalidTestCase = 0;
const UINT_T UnbindTestCase = 2;


static void TC0(void); /* Error */
static void TC1(void); /* Bind */
static void TC2(void); /* Unbind */
static void TC3(void); /* Send */
static void TC4(void); /* Read */

void (*TestCaseFunc[])(void) = { TC0, TC1, TC2, TC3, TC4 };

void signalHandler(int sig);
static void setUp(W16 userId);
static void tearDown(W16 userId);
static void chooseTestCase(INT_T *going);
static void runTestCase(UINT_T number);
static void ShowMenu(void);
int initAppId(char *argv[]);
void beginListening(void);
void arrayCopy(UCHAR_T address[], UCHAR_T nodeXAdr[], int size);
INT_T getTagLen(int msgLen);
```

```
/*=========================================================================*/
/*                F U N C T I O N  D E F I N I T I O N S                    */
/*=========================================================================*/

USHORT_T getTagLen(UINT_T msgLen)
{
   USHORT_T tagLen = 0;

   if( msgLen < 130 )             // 130 == 128 max payload + 2 octets tags
   {
      tagLen = 2;
   }
   else if( msgLen < 258 )        // 258 == 255 max payload + 3 octets tags
   {
      tagLen = 3;
    }
   else                           // Payload < 65536 octets
   {
      tagLen = 4;
    }
    return tagLen;
}


int initAppId(char *argv[])
{
   int inParam = atoi(argv[1]);
    printf("initAppId:\tReceived inparameter: %d\n", inParam);

    switch(inParam){
    case 1:                          /* This is the end node A */
         UserId = USER01_ID;
         Ssn = 20;
         arrayCopy(myNodeAdr, node1Adr, 4);
         break;
    case 2:                          /* This is the SEG node B (SEcurity Gateway)*/
         UserId = USER02_ID;
         Ssn = 19;
         arrayCopy(myNodeAdr, node2Adr, 4);
         break;
    case 3:                          /* This is the SEG node C (SEcurity Gateway)*/
         UserId = USER03_ID;
         Ssn = 9;
         arrayCopy(myNodeAdr, node3Adr, 4);
         break;
    case 4:                          /* This is the end node D */
         UserId = USER04_ID;
         Ssn = 21;
         arrayCopy(myNodeAdr, node4Adr, 4);
         break;
    default:
         printf("initAppId:\tError in inparameter! Choose between 1-4.\n");
         exit(0);
   }

   printf( "initAppId:\tGiven SSN = %d\n", Ssn);
   return inParam;
}


void signalHandler(int sig)
{
   tearDown(UserId);
    exit(sig);
}
```

```
static void setUp(W16 userId)
{
   printf("Init = %d\n", MsgInit(50));
   printf("Open = %d\n", MsgOpen(userId));
   if(MSG_TRACE_ON)
   {
      MsgTraceOn(userId);
      fprintf(stderr, "Message Trace is *On*.\n");
   }
   else
   {
      fprintf(stderr, "Message Trace is *Off*.\n");
   }

   if(XMemInit(UserId, 0, 0, 0) != RETURN_OK)
   {
      printf( "\tXMemInit failed!\n");
      exit(0);
   }
}


static void tearDown(W16 userId)
{
   printf("UnbindReq = %d\n", EINSS7_I97TUnBindReq(Ssn, UserId, 0));
   printf("Rel = %d\n", MsgRel(UserId, TCAP_ID));
   printf("Close = %d\n", MsgClose(userId));
   sleep(1);
   MsgExit();
}


static void chooseTestCase(INT_T *going)
{
   static UCHAR_T string[10];

   printf("\nTestlist(t) quit(q) or test case number: ");
   gets((CHAR_T*)string);

   Testcase = atoi((const CHAR_T*)string);

   if (string[0] == 'q' || string[0] == 'Q')
   {
      *going = 0;
      return;
   }

   if (string[0] == 't' || string[0] == 'T')
   {
      ShowMenu();
      return;
   }

   runTestCase(Testcase);
}

static void runTestCase(UINT_T number)
{
   Running = 1;

   if (number >= sizeof(TestCaseFunc)/sizeof(void (*)(void)))
   {
      number = InvalidTestCase;
   }

   (*TestCaseFunc[number])();

   if ((number == UnbindTestCase) || (number == InvalidTestCase))
   {
      return;
   }
}
```

```
static void ShowMenu(void)
{
    printf(" 1: Bind\n");
    printf(" 2: Unbind\n");
    printf(" 3: Send\n");
    printf(" 4: Read\n");
}


static void TC0(void)
{
    printf("Illegal test case. Please try again.\n");
    Running = 0;
}


static void TC1(void)
{
    USHORT_T result;

    printf("Connect = %d <> TC1\n", MsgConn(UserId, TCAP_ID));

    result = EINSS7_I97TBindReq(Ssn,
                                UserId,
                                0,        /* tcapInstId */
                                1);       /* versionOfTCuser */

    printf("BindReq = %d\n", result);

    while( result != 0)
    {
        printf("Failed to bind to TCAP!\n");
        TC2();
        sleep(2);
        printf("Attempting to rebind\n");
        result = EINSS7_I97TBindReq(Ssn,
                                    UserId,
                                    0,        /* tcapInstId */
                                    1);       /* versionOfTCuser */
        printf("BindReq = %d\n", result);
    }
}


static void TC2(void)
{
   USHORT_T result;

   result = EINSS7_I97TUnBindReq(Ssn, UserId, 0);
    printf("UnbindReq = %d\n", result);

    if (result == 0)
   {
     printf("Rel = %d\n", MsgRel(UserId, TCAP_ID));
    }
    else
   {
     printf("Failed to unbind to TCAP!\n");
    }
    Running = 0;
}
```

```
static void TC3(void)
{
    USHORT_T result = 0;
    UCHAR_T string[10];
    ULONG_T i;

    UCHAR_T destAdr[4];    /* ={ 0x43, 0x34, 0x12, 0x13 };  node2Adr; */
    UCHAR_T orgAdr[4];     /* = { 0x43, 0x34, 0x12, 0x14 };  myNodeAdr; */

    CHAR_T* myMessage = NULL;
    USHORT_T userInfoLength;
    UCHAR_T* userInfo;
    USHORT_T protectionLvl = 4;
    USHORT_T paramLength = 3;
    UCHAR_T* parameters_p;

    SIZE_T msgLen = 0;
    SIZE_T tagLen = 0;
    struct timeval startTime;
    struct timeval stopTime;
    long timeTaken = 0;
    long startMSec = 0;
    long stopMSec = 0;


    printf( "Choose length of message. (0-236)\n");
    gets((char*) string);
    msgLen = atoi((const char*) string);

    while(msgLen > 236)
    {
        printf( "Wrong value!\nChoose length of message. (0-236)\n");
        gets((char*) string);
        msgLen = atoi((const char*) string);
    }

    myMessage = (CHAR_T*)XMalloc(UserId, msgLen);

    tagLen = getTagLen(msgLen);

    debug_printf(( "\n\tTaglength: %u\n\tMessagelength: %u\n", tagLen, msgLen));

    userInfoLength = (USHORT_T)msgLen + (USHORT_T)tagLen;

    /* Allocate length of tag octet + lenght octet(s) + message + \0 char */
    userInfo = (UCHAR_T*)XMalloc(UserId, msgLen + tagLen);

    if(userInfo == NULL)
    {
        printf( "XMalloc Error!!\n");
    }

    userInfo[0] = 0x28;                        /* Insert first tag */

    if(tagLen == 2)
    {
        userInfo[1] = msgLen & 0xFF;
    }
    else if(tagLen == 3)
    {
        userInfo[1] = 0x81;
        userInfo[2] = msgLen & 0xFF;
    }
    else
    {
        userInfo[1] = 0x82;
        userInfo[2] = (msgLen >> 8) & 0xFF;         /* Most significant byte */
        userInfo[3] = msgLen & 0xFF;                /* Least significant byte */
    }
```

```
/* Insert message into allocated memory */
memcpy(&userInfo[tagLen], &myMessage[0], msgLen );

XFree(UserId, myMessage);                          /* Free the memory */

memcpy(destAdr, node2Adr, 4);
memcpy(orgAdr, myNodeAdr, 4);

debug_printf(( "\ttag: 0x%x, length: 0x%x\n", userInfo[0], userInfo[1]));
debug_printf(( "\tuserInfoLength: %u\n", userInfoLength));

printf( "How many dialogues do you want to send in parallell?
    Choose between 1-1000 \n");

gets((char*) string);
noOfParallelDialogues = atoi((const char*) string);

while(noOfParallelDialogues > 1000 || noOfParallelDialogues < 1)
{
    printf( "\nYou have choosen a wrong number of dialogues.\n");
    printf( "How many dialogues do you want to send in parallell?
         Choose between 1-1000 \n");

    gets((char*) string);
    noOfParallelDialogues = atoi((const char*) string);
}

while(protectionLvl > 3)  //Unsigned variable
{
    printf("What level of protection would you like?\n");
    printf("0: Routing through TCAPsec without any security\n");
     printf("1: Routing through TCAPsec with integrity & authentication
         protection\n");
    printf("2: Routing through TCAPsec with full encryption, integrity &
         authentication\n");
    printf("3: Routing directly to end-node without passing TCAPsec SEGs\n");
    printf("Choose between 0-3\n");
    gets((char*) string);
    protectionLvl = atoi((const char*) string);
}

if (protectionLvl == BYPASS_SEG)
{
    memcpy(destAdr, node4Adr, 4);
}

parameters_p = (UCHAR_T*)XMalloc(UserId, paramLength);

if(parameters_p == NULL)
{
    printf( "XMalloc Error!!\n");
}

parameters_p[0] = 0x28;                        /* Insert first tag */
parameters_p[1] = 0x01;                        // The parameter length
parameters_p[2] = protectionLvl & 0xFF;        // The parameter

dialogueID = 1;

// Start timer
result = gettimeofday(&startTime, NULL);
printf( "startTime result %d", result);
```

```
    for( i=0; i<noOfParallelDialogues; i++)
    {
        debug_printf(( "\n\nTC3:\tSending Reqs - DialogID: %d\n", dialogueID));

        do
        {
           result = EINSS7_I97TInvokeReq(Ssn,
                                         UserId,
                                         tcapInstanceId,
                                         dialogueID,
                                         0x01, //invokeId
                                         0x00, //linkedIdUsed
                                         EINSS7_I97TCAP_LINKED_ID_NOT_USED,
                                             //linkedId
                                         EINSS7_I97TCAP_OP_CLASS_1, //opClass
                                         INVOKE_TIMEOUT, //timeout
                                         EINSS7_I97TCAP_OPERATION_TAG_GLOBAL,
                                             //operationTag
                                         (msgLen + tagLen), //operationLength
                                         userInfo, //*operationCode_p
                                         paramLength, //paramLength
                                         parameters_p); // *parameters_p
            if(result != 0)
            {
                printf( "Failed InvokeReq %d\n", result);
            }
        }while(result != 0);

        /* destAdr is the address of Node 2. */
        do
        {
          result = EINSS7_I97TBeginReq( Ssn,
                                         UserId,
                                         tcapInstanceId,
                                         dialogueID,
                                         priOrder,
                                         qualityOfService,
                                         destAdrLength,
                                         &destAdr[0],
                                         orgAdrLength,
                                         &orgAdr[0],
                                         appContextLength,
                                         appContext_p,
                                         0, //userInfoLength,
                                         NULL);  //userInfo);
          if(result != 0)
          {
              printf( "Failed BeginReq %d\n", result);
          }
       }while(result != 0);

       dialogueID++;

       EINSS7CpXSleepMilli(1);      // Set at least 1 millisecond sleep between each
                                    // sending to avoid congestion
    }

    XFree(UserId, userInfo);       /* Free the memory */

    /* Every message is sent and the node starts listening */
    while( noOfParallelDialogues != 0 )
    {
        beginListening();
    }

    // Stop timer and print elapsed time
    gettimeofday(&stopTime, NULL);
    startMSec = (startTime.tv_sec * 1000) + (startTime.tv_usec / 1000);
    stopMSec = (stopTime.tv_sec * 1000) + (stopTime.tv_usec / 1000);
    timeTaken = stopMSec - startMSec;
    printf( "\n\tElapsed time in milliseconds: %ld\n", timeTaken);
}
```

```
static void TC4(void)
{
    while(1)
    {
        beginListening();
    }
}


void beginListening(void)
{
    static UCHAR_T buf[400];
    USHORT_T result = 0;

    MSG_T msg_s;
    SHORT_T timeout = MSG_INFTIM;

    msg_s.msg_p = &buf[0];
    msg_s.receiver = UserId;

    result = MsgRecvEvent(&msg_s, NULL, NULL, timeout);

    if (result == MSG_ERR)
    {
        puts("System Error! Exiting...");
        tearDown(UserId);
        exit(result);
    }

    if (msg_s.sender != TCAP_ID)
    {
        debug_printf( ("Dumping incoming message from %d\n", msg_s.sender));
    }
    else
    {
        result = EINSS7_I97THandleInd(&msg_s);

        if (result != 0)
        {
            debug_printf(("Error in calling T_Handle_ind from beginListening()
                - result =  %d\n", result));
        }
    }
}


void arrayCopy(UCHAR_T address[], UCHAR_T nodeXAdr[], int size)
{
    int i;
    for(i = 0; i < size; i++)
    {
        address[i] = nodeXAdr[i];
    }
}


USHORT_T EINSS7_I97TbindConf(
    UCHAR_T ssn,
    USHORT_T userid,
    EINSS7INSTANCE_T tcapInstanceId_q,
    UCHAR_T result)
{
    tcapInstanceId_q = tcapInstanceId_q; /* To remove warnings */

    printf(" Received: T_BIND_conf\n");
    printf(" SSN = %d\n", ssn);
    printf(" User ID = %d\n", userid);
    printf(" Result = %d\n", result);
```

```
    if (result != 0)
    {
        printf("Failed to bind to TCAP!\n");
        Bound = 0;
    }
    else
    {
        printf("Success!\n");
        Bound = 1;
    }

    Running = 0;
    return result;
}

USHORT_T EINSS7_I97TRRejectInd(
    UCHAR_T localSsn_p,
    USHORT_T userid_p,
    EINSS7INSTANCE_T tcapInstanceId_p,
    USHORT_T dialogueId_p,
    UCHAR_T invokeIdUsed_p,
    UCHAR_T invokeId_p,
    UCHAR_T lastComponent_p,
    UCHAR_T problemCodeTag_p,
    UCHAR_T problemCode_p)
{
    (void)localSsn_p;
    (void)userid_p;
    (void)tcapInstanceId_p;
    (void)dialogueId_p;
    (void)invokeIdUsed_p;
    (void)invokeId_p;
    (void)lastComponent_p;
    (void)problemCodeTag_p;
    (void)problemCode_p;

    printf("Received: T_R_REJECT_ind\n");
    return 0;
}

USHORT_T EINSS7_I97TURejectInd(
    UCHAR_T /*localSsn_o*/,
    USHORT_T /*userid_o*/,
    EINSS7INSTANCE_T /*tcapInstanceId_o*/,
    USHORT_T /*dialogueId_o*/,
    UCHAR_T /*invokeIdUsed_o*/,
    UCHAR_T /*invokeId_o*/,
    UCHAR_T /*lastComponent_o*/,
    UCHAR_T /*problemCodeTag_o*/,
    UCHAR_T /*problemCode_o*/)
{
    printf("Received: T_U_REJECT_ind\n");
    return 0;
}

USHORT_T EINSS7_I97TStateInd(
    UCHAR_T /*localSsn_n*/,
    USHORT_T /*userid_n*/,
    EINSS7INSTANCE_T /*tcapInstanceId_n*/,
    UCHAR_T userState_n,
    UCHAR_T affectedSsn_n,
    ULONG_T affectedSpc_n,
    ULONG_T /*localSpc_n*/,
    UCHAR_T /*subsysMultiplicityInd_n*/ )
{

    if(affectedSpc_n == 4660 || affectedSpc_n == 839)
    {
        printf("AffectedSPC: %lu\n", affectedSpc_n);
```

```
        if(affectedSsn_n == 9 || affectedSsn_n == 19 || affectedSsn_n == 20 ||
             affectedSsn_n == 21)
        {
            printf("Userstate: %u\n", userState_n);
            printf("Affected SSN: %u\n", affectedSsn_n);
        }
    }

    printf("\t\t ---- Received T_State_Ind ----\n");
    return 0;
}


USHORT_T EINSS7_I97TPAbortInd(UCHAR_T localSsn_m,
                              USHORT_T userid_m,
                              EINSS7INSTANCE_T tcapInstanceId_m,
                              USHORT_T dialogueId_m,
                              UCHAR_T priOrder_m,
                              UCHAR_T qos_m,
                              UCHAR_T abortCause_m)
{
    printf("\tReceived T_P_Abort_ind\n");
    printf("localSsn_m %d\n", localSsn_m);
    printf("userid_m %d\n", userid_m);
    printf("tcapInstanceId_m %d\n", tcapInstanceId_m);
    printf("dialogueId_m %d\n", dialogueId_m);
    printf("priOrder_m %d\n", priOrder_m);
    printf("qos_m %d\n", qos_m);
    printf(" Abort Cause = %d\n\n", abortCause_m);
    printf("\tdialogueId_m %d\n", dialogueId_m);
    printf("\tAbort Cause = %d\n\n", abortCause_m);
    return 0;
}

USHORT_T EINSS7_I97TLRejectInd(UCHAR_T localSsn_l,
                               USHORT_T userid_l,
                               EINSS7INSTANCE_T tcapInstanceId_l,
                               USHORT_T dialogueId_l,
                               UCHAR_T invokeIdUsed_l,
                               UCHAR_T invokeId_l,
                               UCHAR_T problemCodeTag_l,
                               UCHAR_T problemCode_l)
{
    printf("Received: T_L_REJECT_ind\n");
    printf("localSsn_l %d\n", localSsn_l);
    printf("userid_l %d\n", userid_l);
    printf("tcapInstanceId_l %d\n", tcapInstanceId_l);
    printf("dialogueId_l %d\n", dialogueId_l);
    printf("invokeIdUsed_l %d\n", invokeIdUsed_l);
    printf("invokeId_l %d\n", invokeId_l);
    printf("\n");
    printf(" Problem Code Tag = %d\n", problemCodeTag_l);
    printf(" Problem Code = %d\n", problemCode_l);
    return 0;
}

USHORT_T EINSS7_I97TResultNLInd(
    UCHAR_T localSsn_k,
    USHORT_T userid_k,
    EINSS7INSTANCE_T tcapInstanceId_k,
    USHORT_T dialogueId_k,
    UCHAR_T invokeId_k,
    UCHAR_T lastComponent_k,
    UCHAR_T operationTag_k,
    USHORT_T operationLength_k,
    UCHAR_T *operationCode_p_k,
    USHORT_T paramLength_k,
    UCHAR_T *parameters_p_k)
{
    printf("localSsn_k %c\n", localSsn_k);
    printf("userid_k %d\n", userid_k);
    printf("tcapInstanceId_k %d\n", tcapInstanceId_k);
```

```
    printf("dialogueId_k %d\n", dialogueId_k);
    printf("invokeId_k %c\n", invokeId_k);
    printf("lastComponent_k %c\n",  lastComponent_k);
    printf("operationTag_k %c\n", operationTag_k);
    printf("operationLength_k %d\n", operationLength_k);
    printf("*operationCode_p_k %c\n", *operationCode_p_k);
    printf("paramLength_k %d\n", paramLength_k);
    printf("*parameters_p_k %c\n", *parameters_p_k);
    printf("\n");
    printf("Received: T_RESULT_NL_ind\n");
    return 0;
}

USHORT_T EINSS7_I97TResultLInd(
    UCHAR_T localSsn_b,
    USHORT_T userid_b,
    EINSS7INSTANCE_T tcapInstanceId_b,
    USHORT_T dialogueId_b,
    UCHAR_T invokeId_b,
    UCHAR_T lastComponent_b,
    UCHAR_T operationTag_b,
    USHORT_T operationLength_b,
    UCHAR_T *operationCode_p_b,
    USHORT_T paramLength_b,
    UCHAR_T *parameters_p_b)
{
    USHORT_T result;
    UCHAR_T destAdr[4];
    UINT_T tagLen = 0;
    UCHAR_T* newOperationCode_p = NULL;

    (void)userid_b;
    (void)lastComponent_b;
    (void)paramLength_b;

    if (Ssn == 20)
    {
        printf("\tReceived:; T_RESULT_L_ind\tInvokes: %d\tResults: %d\n", invokes,
            ++results);
    }
    else
    {
        debug_printf(("\tReceived:; T_RESULT_L_ind\tInvokes: %d\tResults: %d\n",
            invokes, ++results));
    }

    debug_printf(("operationLength_b %d\n", operationLength_b));

    if(Ssn == 20 || Ssn == 21)
    {
        debug_printf(( "Message received. Length: %d\n", operationLength_b));
    }

    debug_printf(("Protectionlvl = %u\n", parameters_p_b[2]));

    if( operationLength_b < 130 )// 130 == 128 max payload + 2 octets tags
    {
        tagLen = 2;
    }
    else if( operationLength_b < 258 )  // 258 == 255 max payload + 3 octets tags
    {
        tagLen = 3;
    }
    else                                // Payload < 65536 octets
    {
        tagLen = 4;
    }
```

```
/* Generate key and IV*/
 std::string passphrase = PASSPHRASE;

 /* Initiate encryption pipes */
 if((Ssn == 19 && dialogueId_b > 30000) ||
    (Ssn == 9 && dialogueId_b < 30000))      // ** Outgoing msgs **
{
      if(parameters_p_b[2] == 1)
      {
         Pipe protPipe(get_cipher(INTEGR_ALGOR, *key, *iv, ENCRYPTION));

         debug_printf(( "\t**Performing Encryption lvl 1\n"));

         protPipe.process_msg(operationCode_p_b, operationLength_b);

         const u32bit expecting = protPipe.remaining();

         // byte == Botan's typedef for unsigned char
         byte* output = new byte[expecting];
         protPipe.read(output, expecting);

         tagLen = getTagLen(expecting);

         newOperationCode_p = (UCHAR_T*)XMalloc(UserId, expecting + tagLen);
         newOperationCode_p[0] = 0x28;

         if(tagLen == 2)
         {
             newOperationCode_p[1] = expecting & 0xFF;
         }
        else if(tagLen == 3)
         {
             newOperationCode_p[1] = 0x81;
             newOperationCode_p[2] = expecting & 0xFF;
         }
        else
         {
             newOperationCode_p[1] = 0x82;
             /* Most significant byte */
             newOperationCode_p[2] = (expecting >> 8) & 0xFF;
             /* Least significant byte */
             newOperationCode_p[3] = expecting & 0xFF;
         }

         memcpy(&newOperationCode_p[tagLen], output, expecting);
         operationLength_b = expecting + tagLen;

      }
      else if(parameters_p_b[2] == 2)
      {
         Pipe protPipe(get_cipher(CRYPTO_ALGOR, *key, *iv, ENCRYPTION),
                       get_cipher(INTEGR_ALGOR, *key, *iv, ENCRYPTION));

         debug_printf(( "\t**Performing Encryption lvl 2\n"));

         protPipe.process_msg(operationCode_p_b, operationLength_b);
         const u32bit expecting = protPipe.remaining();

         // byte == Botan's typedef for unsigned char
         byte* output = new byte[expecting];
         protPipe.read(output, expecting);

         tagLen = getTagLen(expecting);

         newOperationCode_p = (UCHAR_T*)XMalloc(UserId, expecting + tagLen);
         newOperationCode_p[0] = 0x28;

        if(tagLen == 2)
         {
             newOperationCode_p[1] = expecting & 0xFF;
         }
```

```
        else if(tagLen == 3)
        {
            newOperationCode_p[1] = 0x81;
            newOperationCode_p[2] = expecting & 0xFF;
        }
        else
        {
            newOperationCode_p[1] = 0x82;
            /* Most significant byte */
            newOperationCode_p[2] = (expecting >> 8) & 0xFF;
            /* Least significant byte */
            newOperationCode_p[3] = expecting & 0xFF;
        }
        memcpy(&newOperationCode_p[tagLen], output, expecting);
        operationLength_b = expecting + tagLen;
     }
     else
     {
        /* no protection*/
        newOperationCode_p = operationCode_p_b;
        debug_printf(( "\t**No protection performed\n"));
     }
}
else if((Ssn == 19 && dialogueId_b < 30000) ||
    (Ssn == 9 && dialogueId_b > 30000))          // ** Incoming msgs **
{
    if(parameters_p_b[2] == 1)
    {
        Pipe protPipe(get_cipher(INTEGR_ALGOR, *key, *iv, DECRYPTION));

        debug_printf(( "\t**Performing Decryption lvl 1\n"));

        protPipe.process_msg(&operationCode_p_b[tagLen], operationLength_b
            - tagLen);

        const u32bit expecting = protPipe.remaining();

        // byte == Botan's typedef for unsigned char
        byte* output = new byte[expecting];
        protPipe.read(output, expecting);

        tagLen = getTagLen(expecting);

        newOperationCode_p = (UCHAR_T*)XMalloc(UserId, expecting);
        memcpy(&newOperationCode_p[0], output, expecting);
        operationLength_b = expecting;

    }
    else if(parameters_p_b[2] == 2)
    {
       Pipe protPipe(get_cipher(INTEGR_ALGOR, *key, *iv, DECRYPTION),
            get_cipher(CRYPTO_ALGOR, *key, *iv, DECRYPTION));

       debug_printf(( "\t**Performing Decryption lvl 2\n"));

       protPipe.process_msg(&operationCode_p_b[tagLen], operationLength_b
            - tagLen);
        const u32bit expecting = protPipe.remaining();

        // byte == Botan's typedef for unsigned char
        byte* output = new byte[expecting];
        protPipe.read(output, expecting);

        tagLen = getTagLen(expecting);

        newOperationCode_p = (UCHAR_T*)XMalloc(UserId, expecting + tagLen);
        memcpy(&newOperationCode_p[0], output, expecting);
        operationLength_b = expecting;
    }
```

```
    else
    {
        newOperationCode_p = operationCode_p_b;
        debug_printf(( "\t**No protection performed\n"));
    }
}

if(localSsn_b == 19)
{
    if(dialogueId_b > 30000)
    {
        dialogueId_b += 2000;
        memcpy(destAdr, node3Adr, 4);
    }
    else
    {
        dialogueId_b -= 2000;
        memcpy(destAdr, node1Adr, 4);
    }
}
else if(localSsn_b == 9)
{
    if(dialogueId_b > 30000)
    {
        dialogueId_b += 2000;
        memcpy(destAdr, node4Adr, 4);
    }
    else
    {
        dialogueId_b -= 2000;
        memcpy(destAdr, node2Adr, 4);
    }
}
else if(localSsn_b == 21)
{
    memcpy(destAdr, node3Adr, 4);
    newOperationCode_p = operationCode_p_b;
}
else if(Ssn == 20)
{
    /* Reached the beginning */
}
else
{
    printf( "Wrong receiving SSN!!\n");
}

debug_printf(("--operationLength after processing: %d\n", operationLength_b));
debug_printf(("New dialogueId: %u\n", dialogueId_b));

if(Ssn == 19 || Ssn == 9 || Ssn == 21)
{
    if((Ssn == 19 && dialogueId_b > 30000) ||
       (Ssn == 9 && dialogueId_b > 30000) ||
       Ssn == 21)
    {
      do
      {
         result = EINSS7_I97TResultLReq(Ssn,
                                        UserId,
                                        tcapInstanceId_b,
                                        dialogueId_b,
                                        invokeId_b,
                                        operationTag_b,
                                        operationLength_b,
                                        newOperationCode_p,
                                        paramLength_b,
                                        &parameters_p_b[0]);

         debug_printf(("ResultReq sent, dialogueId: %u\n", dialogueId_b));
```

```
            if(result != 0)
            {
               printf("Failed ResultReq: %d\n", result);
            }
         }
         while(result != 0);
      }

      if(dialogueId_b > 30000)
      {
         do
         {
            result = EINSS7_I97TEndReq(Ssn,
                                       UserId,
                                       tcapInstanceId_b,
                                       dialogueId_b,
                                       priOrder,
                                       qualityOfService,
                                       EINSS7_I97TCAP_TERM_BASIC_END,
                                       appContextLength,
                                       appContext_p,
                                       0,      //userInfoLength
                                       NULL);  //userInfo

            if(result != 0)
            {
               printf("Failed EndReq: %d\n", result);
            }

         }while(result != 0);

         debug_printf(("EndReq sent, dialogueId: %u\n", dialogueId_b));
      }
      else if(dialogueId_b > 0)
      {
         do
         {
             result = EINSS7_I97TBeginReq(Ssn,
                                          UserId,
                                          tcapInstanceId,
                                          dialogueId_b,
                                          priOrder,
                                          qualityOfService,
                                          destAdrLength,
                                          &destAdr[0],
                                          orgAdrLength,
                                          &myNodeAdr[0],
                                          appContextLength,
                                          appContext_p,
                                          0,      //userInfoLength
                                          NULL);  //userInfo
             if(result != 0)
             {
               printf("Failed BeginReq: %d\n", result);
             }
         }while(result != 0);

         debug_printf(("BeginReq sent, dialogueId: %u\n", dialogueId_b));
      }
   }
   else
   {
      noOfParallelDialogues--;         /* Count down active dialogues */
   }
   if((Ssn == 19 || Ssn == 9) && (parameters_p_b[2] > 0))
   {
      XFree(UserId, newOperationCode_p);
   }

   return 0;
}
```

```
USHORT_T EINSS7_I97TNoticeInd(
    UCHAR_T localSsn_i,
    USHORT_T userid_i,
    EINSS7INSTANCE_T tcapInstanceId_i,
    USHORT_T dialogueId_i,
    UCHAR_T reportCause_i,
    UCHAR_T returnIndicator_i,
    USHORT_T relDialogueId_i,
    UCHAR_T segmInd_i,
    UCHAR_T destAdrLength_i,
    UCHAR_T *destAdr_p_i,
    UCHAR_T orgAdrLength_i,
    UCHAR_T *orgAdr_p_i)
{
    printf("Received: T_NOTICE_ind\n");
    printf("localSsn_i %c\n", localSsn_i);
    printf("userid_i %d\n", userid_i);
    printf("tcapInstanceId_i %d\n", tcapInstanceId_i);
    printf("dialogueId_i %d\n", dialogueId_i);
    printf("reportCause_i %c\n", reportCause_i);
    printf("returnIndicator_i %c\n", returnIndicator_i);
    printf("relDialogueId_i %d\n", relDialogueId_i);
    printf("segmInd_i %c\n", segmInd_i);
    printf("destAdrLength_i %c\n", destAdrLength_i);
    printf(" *destAdr_p_i %c\n", *destAdr_p_i);
    printf("orgAdrLength_i %c\n", orgAdrLength_i);
    printf("*orgAdr_p_i %c\n", *orgAdr_p_i);
    printf("\n");

    return 0;
}

USHORT_T EINSS7_I97TUErrorInd(
    UCHAR_T localSsn_h,
    USHORT_T userid_h,
    EINSS7INSTANCE_T tcapInstanceId_h,
    USHORT_T dialogueId_h,
    UCHAR_T invokeId_h,
    UCHAR_T lastComponent_h,
    UCHAR_T errorCodeTag_h,
    USHORT_T errorCodeLength_h,
    UCHAR_T *errorCode_p_h,
    USHORT_T paramLength_h,
    UCHAR_T *parameters_p_h)
{
    printf("Received: T_U_ERROR_ind\n");
    printf("localSsn_h %c\n", localSsn_h);
    printf("userid_h %d\n", userid_h);
    printf("tcapInstanceId_h %d\n", tcapInstanceId_h);
    printf("dialogueId_h %d\n", dialogueId_h);
    printf("invokeId_h %c\n", invokeId_h);
    printf("lastComponent_h %c\n", lastComponent_h);
    printf("errorCodeTag_h %c\n", errorCodeTag_h);
    printf("errorCodeLength_h %d\n", errorCodeLength_h);
    printf("*errorCode_p_h %c\n", *errorCode_p_h);
    printf("paramLength_h %d\n", paramLength_h);
    printf("*parameters_p_h %c\n", *parameters_p_h);
    printf("\n");

    printf(" Error Code = %d\n", *errorCode_p_h);
    return 0;
}

USHORT_T EINSS7_I97TLCancelInd(
    UCHAR_T localSsn_g,
    USHORT_T userid_g,
    EINSS7INSTANCE_T tcapInstanceId_g,
    USHORT_T dialogueId_g,
    UCHAR_T invokeId_g)
{
    printf("Received: T_L_CANCEL_ind\n");
    printf("localSsn %c\n", localSsn_g);
```

```
    printf("userid %d\n", userid_g);
    printf("tcapInstanceId %d\n", tcapInstanceId_g);
    printf(" dialogueId %d\n", dialogueId_g);
    printf("invokeId %c\n", invokeId_g);
    printf("\n");

    return 0;
}

USHORT_T EINSS7_I97TUAbortInd(
    UCHAR_T localSsn_f,
    USHORT_T userid_f,
    EINSS7INSTANCE_T tcapInstanceId_f,
    USHORT_T dialogueId_f,
    UCHAR_T priOrder_f,
    UCHAR_T qualityOfService_f,
    USHORT_T abortInfoLength_f,
    UCHAR_T *abortInfo_p_f,
    UCHAR_T appContextLength_f,
    UCHAR_T *appContext_p_f,
    USHORT_T userInfoLength_f,
    UCHAR_T *userInfo_p_f)
{
    printf("Received: T_U_ABORT_ind\n");
    printf("localSsn %c\n", localSsn_f);
    printf("userid %d\n", userid_f);
    printf("tcapInstanceId %d\n", tcapInstanceId_f);
    printf("dialogueId %d\n", dialogueId_f);
    printf("priOrder %c\n", priOrder_f);
    printf("qualityOfService %c\n", qualityOfService_f);
    printf("abortInfoLength %d\n", abortInfoLength_f);
    printf("*abortInfo_p %c\n", *abortInfo_p_f);
    printf("appContextLength %c\n", appContextLength_f);
    printf("*appContext_p %c\n", *appContext_p_f);
    printf("userInfoLength %d\n", userInfoLength_f);
    printf("*userInfo_p %c\n",*userInfo_p_f);
    printf("\n");

    return 0;
}

USHORT_T EINSS7_I97TIndError(
    USHORT_T  indLenErr,
    MSG_T *mqp_sp)
{

    printf("indLenErr %d\n", indLenErr);
    (void)mqp_sp;                           /* to remove warning */
    printf("\n");

    printf("Received: TIndError\n");
    return 0;
}

USHORT_T EINSS7_I97TContinueInd(
    UCHAR_T localSsn_e,
    USHORT_T userid_e,
    EINSS7INSTANCE_T tcapInstanceId_e,
    USHORT_T dialogueId_e,
    UCHAR_T priOrder_e,
    UCHAR_T qualityOfService_e,
    UCHAR_T compPresent_e,
    UCHAR_T appContextLength_e,
    UCHAR_T *appContext_p_e,
    USHORT_T userInfoLength_e,
    UCHAR_T *userInfo_p_e)
{

    printf("localSsn %c\n", localSsn_e);
    printf("userid %d\n", userid_e);
    printf("tcapInstanceId %d\n", tcapInstanceId_e);
    printf("dialogueId %d\n", dialogueId_e);
```

```
    printf("priOrder %c\n", priOrder_e);
    printf("qualityOfService %c\n", qualityOfService_e);
    printf("compPresent %c\n", compPresent_e);
    printf("appContextLength %c\n", appContextLength_e);
    printf("*appContext_p %c\n", *appContext_p_e);
    printf("userInfoLength %d\n", userInfoLength_e);
    printf("*userInfo_p %c\n",*userInfo_p_e);
    printf("\n");

    printf("Received T_CONTINUE_ind\n");
    return 0;
}

USHORT_T EINSS7_I97TUniInd(
    UCHAR_T localSsn_d,
    USHORT_T userid_d,
    EINSS7INSTANCE_T tcapInstanceId_d,
    UCHAR_T priOrder_d,
    UCHAR_T qos_d,
    UCHAR_T destAdrLength_d,
    UCHAR_T *destAdr_p_d,
    UCHAR_T orgAdrLength_d,
    UCHAR_T *orgAdr_p_d,
    UCHAR_T compPresent_d,
    UCHAR_T appContextLength_d,
    UCHAR_T *appContext_p_d,
    USHORT_T userInfoLength_d,
    UCHAR_T *userInfo_p_d)
{
    printf("localSsn %c\n", localSsn_d);
    printf("userid %d\n", userid_d);
    printf("tcapInstanceId %d\n", tcapInstanceId_d);
    printf("priOrder %c\n", priOrder_d);
    printf("qos %c\n", qos_d);
    printf("destAdrLength %c\n", destAdrLength_d);
    printf("*destAdr_p %c\n", *destAdr_p_d);
    printf("orgAdrLength %c\n", orgAdrLength_d);
    printf("*orgAdr_p %c\n", *orgAdr_p_d);
    printf("compPresent %c\n", compPresent_d);
    printf("appContextLength %c\n", appContextLength_d);
    printf("*appContext_p %c\n", *appContext_p_d);
    printf("userInfoLength %d\n", userInfoLength_d);
    printf("*userInfo_p %c\n",*userInfo_p_d);
    printf("\n");


    printf("Received: T_UNI_ind\n");
    return 0;
}

USHORT_T EINSS7_I97TBeginInd(
    UCHAR_T localSsn_c,
    USHORT_T userid_c,
    EINSS7INSTANCE_T tcapInstanceId_c,
    USHORT_T dialogueId_c,
    UCHAR_T priOrder_c,
    UCHAR_T qualityOfService_c,
    UCHAR_T destAdrLength_c,
    UCHAR_T *destAdr_p_c,
    UCHAR_T orgAdrLength_c,
    UCHAR_T *orgAdr_p_c,
    UCHAR_T compPresent_c,
    UCHAR_T appContextLength_c,
    UCHAR_T *appContext_p_c,
    USHORT_T userInfoLength_c,
    UCHAR_T *userInfo_p_c)
{
    debug_printf(("\n\tReceived T_Begin_Req\n"));
    debug_printf(("\tdialogueId: %d\n\n", dialogueId_c));

    (void)localSsn_c;
    (void)userid_c;
```

```
    (void)tcapInstanceId_c;
    (void)dialogueId_c;
    (void)priOrder_c;
    (void)qualityOfService_c;
    (void)destAdrLength_c;
    (void)destAdr_p_c;
    (void)orgAdrLength_c;
    (void)orgAdr_p_c;
    (void)compPresent_c;
    (void)appContextLength_c;
    (void)appContext_p_c;
    (void)userInfoLength_c;
    (void)*userInfo_p_c;

    return 0;
}

USHORT_T EINSS7_I97TInvokeInd(UCHAR_T localSsn_b,
                              USHORT_T userid_b,
                              EINSS7INSTANCE_T tcapInstanceId_b,
                              USHORT_T dialogueId_b,
                              UCHAR_T invokeId_b,
                              UCHAR_T lastComponent_b,
                              UCHAR_T linkedIdUsed_b,
                              UCHAR_T linkedId_b,
                              UCHAR_T operationTag_b,
                              USHORT_T operationLength_b,
                              UCHAR_T *operationCode_p_b,
                              USHORT_T paramLength_b,
                              UCHAR_T *parameters_p_b)
{
    USHORT_T result;
    UCHAR_T destAdr[4];
    UINT_T tagLen = 0;
    UCHAR_T* newOperationCode_p = NULL;


    (void)userid_b;
    (void)lastComponent_b;
    (void)paramLength_b;

    if (Ssn == 21)
    {
        printf("\tReceived: T_INVOKE_ind\tInvokes: %d\tResults: %d\n", ++invokes,
           results);
    }
    else
    {
        debug_printf(("\tReceived: T_INVOKE_ind\tInvokes: %d\tResults: %d\n",
           ++invokes, results));
    }

    debug_printf(("operationLength_b %d\n", operationLength_b));

    if(Ssn == 20 || Ssn == 21)
    {
        debug_printf(( "Message received\n"));
    }

    debug_printf(("Protectionlvl = %u\n", parameters_p_b[2]));


    if( operationLength_b < 130 ) //130 == 128 max payload + 2 octets tags
    {
        tagLen = 2;
    }
    else if( operationLength_b < 258 ) //258 == 255 max payload + 3octets tags
    {
        tagLen = 3;
    }
```

```
else  // Payload < 65536 octets
{
    tagLen = 4;
}

/* Initiate encryption pipes */

if((Ssn == 19 && dialogueId_b > 30000) ||
   (Ssn == 9 && dialogueId_b < 30000))       // ** Outgoing msgs **
{
    if(parameters_p_b[2] == 1)
    {
        Pipe protPipe(get_cipher(INTEGR_ALGOR, *key, *iv, ENCRYPTION));

        debug_printf(( "\t**Performing Encryption lvl 1\n"));

        protPipe.process_msg(operationCode_p_b, operationLength_b);

        const u32bit expecting = protPipe.remaining();
        // byte == Botan's typedef for unsigned char
        byte* output = new byte[expecting];
        protPipe.read(output, expecting);

        tagLen = getTagLen(expecting);

        newOperationCode_p = (UCHAR_T*)XMalloc(UserId, expecting + tagLen);
        newOperationCode_p[0] = 0x28;

        if(tagLen == 2)
        {
            newOperationCode_p[1] = expecting & 0xFF;
        }
        else if(tagLen == 3)
        {
            newOperationCode_p[1] = 0x81;
            newOperationCode_p[2] = expecting & 0xFF;
        }
        else
        {
            newOperationCode_p[1] = 0x82;
                 /* Most significant byte */
            newOperationCode_p[2] = (expecting >> 8) & 0xFF;
                 /* Least significant byte */
            newOperationCode_p[3] = expecting & 0xFF;
        }

        memcpy(&newOperationCode_p[tagLen], output, expecting);
        operationLength_b = expecting + tagLen;

    }
    else if(parameters_p_b[2] == 2)
    {
        Pipe protPipe(get_cipher(CRYPTO_ALGOR, *key, *iv, ENCRYPTION),
                      get_cipher(INTEGR_ALGOR, *key, *iv, ENCRYPTION));

        debug_printf(( "\t**Performing Encryption lvl 2\n"));

        protPipe.process_msg(operationCode_p_b, operationLength_b);
        const u32bit expecting = protPipe.remaining();

        // byte == Botan's typedef for unsigned char
        byte* output = new byte[expecting];
        protPipe.read(output, expecting);

        tagLen = getTagLen(expecting);

        newOperationCode_p = (UCHAR_T*)XMalloc(UserId, expecting + tagLen);
        newOperationCode_p[0] = 0x28;
```

```
        if(tagLen == 2)
        {
            newOperationCode_p[1] = expecting & 0xFF;
        }
        else if(tagLen == 3)
        {
            newOperationCode_p[1] = 0x81;
            newOperationCode_p[2] = expecting & 0xFF;
        }
        else
        {
            newOperationCode_p[1] = 0x82;
            /* Most significant byte */
            newOperationCode_p[2] = (expecting >> 8) & 0xFF;
            /* Least significant byte */
            newOperationCode_p[3] = expecting & 0xFF;
        }
        memcpy(&newOperationCode_p[tagLen], output, expecting);
        operationLength_b = expecting + tagLen;
    }
    else
    {
        /* no protection*/
        newOperationCode_p = operationCode_p_b;
        debug_printf(( "\t**No protection performed\n"));
    }
}
else if((Ssn == 19 && dialogueId_b < 30000) ||
    (Ssn == 9 && dialogueId_b > 30000))     // ** Incoming msgs **
{
    if(parameters_p_b[2] == 1)
    {
        Pipe protPipe(get_cipher(INTEGR_ALGOR, *key, *iv, DECRYPTION));

        debug_printf(( "\t**Performing Decryption lvl 1\n"));

        protPipe.process_msg(&operationCode_p_b[tagLen], operationLength_b
            - tagLen);

        const u32bit expecting = protPipe.remaining();
        // byte == Botan's typedef for unsigned char
        byte* output = new byte[expecting];
        protPipe.read(output, expecting);

        tagLen = getTagLen(expecting);

        newOperationCode_p = (UCHAR_T*)XMalloc(UserId, expecting);

        memcpy(&newOperationCode_p[0], output, expecting);
        operationLength_b = expecting;
    }
    else if(parameters_p_b[2] == 2)
    {
        Pipe protPipe(get_cipher(INTEGR_ALGOR, *key, *iv, DECRYPTION),
                      get_cipher(CRYPTO_ALGOR, *key, *iv, DECRYPTION));

        debug_printf(( "\t**Performing Decryption lvl 2\n"));

        protPipe.process_msg(&operationCode_p_b[tagLen], operationLength_b
            - tagLen);

        const u32bit expecting = protPipe.remaining();
        // byte == Botan's typedef for unsigned char
        byte* output = new byte[expecting];
        protPipe.read(output, expecting);

        tagLen = getTagLen(expecting);

        newOperationCode_p = (UCHAR_T*)XMalloc(UserId, expecting + tagLen);
        memcpy(&newOperationCode_p[0], output, expecting);
        operationLength_b = expecting;
    }
```

```
    else
    {
        /* no protection*/
        newOperationCode_p = operationCode_p_b;
        debug_printf(( "\t**No protection performed\n"));
    }
}

if(localSsn_b == 19)
{
    if(dialogueId_b > 30000)
    {
        dialogueId_b += 2000;
        memcpy(destAdr, node3Adr, 4);
    }
    else
    {
        dialogueId_b -= 2000;
        memcpy(destAdr, node1Adr, 4);
    }
}
else if(localSsn_b == 9)
{
    if(dialogueId_b > 30000)
    {
        dialogueId_b += 2000;
        memcpy(destAdr, node4Adr, 4);
    }
    else
    {
        dialogueId_b -= 2000;
        memcpy(destAdr, node2Adr, 4);
    }
}
else if(localSsn_b == 21)
{
    if (parameters_p_b[2] == BYPASS_SEG)
    {
        memcpy(destAdr, node4Adr, 4);
    }
    else
    {
        memcpy(destAdr, node3Adr, 4);
    }
    newOperationCode_p = operationCode_p_b;
}
else if(Ssn == 20)
{
    /* Reached the beginning */
}
else
{
    printf( "Wrong receiving SSN!!\n");
}

debug_printf(("--operationLength after processing: %d\n", operationLength_b));

if(Ssn == 19 || Ssn == 9 || Ssn == 21)
{
    if((Ssn == 19 && dialogueId_b < 30000)||(Ssn == 9 && dialogueId_b < 30000))
    {
      do
      {
         result = EINSS7_I97TInvokeReq(Ssn,
                                       UserId,
                                       tcapInstanceId_b,
                                       dialogueId_b,
                                       invokeId_b,//0x01, //invokeId
                                       linkedIdUsed_b, //linkedIdUsed
                                       linkedId_b, //linkedId
                                       EINSS7_I97TCAP_OP_CLASS_1, //opClass
                                       INVOKE_TIMEOUT, //timeout
```

```
                                    operationTag_b, //operationTag
                                    operationLength_b, //operationLength
                                    newOperationCode_p, // *operationCode_p
                                    paramLength_b, //paramLength
                                    &parameters_p_b[0]); // *parameters_p

      if(result != 0)
      {
         printf("Failed InvokeReq: %d\n", result);
      }
   }while(result != 0);

    debug_printf(("InvokeReq sent, dialogueId: %u\n", dialogueId_b));
}
else if((Ssn == 19 && dialogueId_b > 30000) ||
      (Ssn == 9 && dialogueId_b > 30000) || Ssn == 21)
{
   do
   {
      result = EINSS7_I97TResultLReq(Ssn,
                                     UserId,
                                     tcapInstanceId_b,
                                     dialogueId_b,
                                     invokeId_b,
                                     operationTag_b,
                                     operationLength_b,
                                     newOperationCode_p,
                                     paramLength_b,
                                     &parameters_p_b[0]);
      if(result != 0)
      {
         printf("Failed ResultReq: %d\n", result);
      }
   }while(result != 0);

}

if(dialogueId_b > 30000)
{
   do
   {
      result = EINSS7_I97TEndReq(Ssn,
                                 UserId,
                                 tcapInstanceId_b,
                                 dialogueId_b,
                                 priOrder,
                                 qualityOfService,
                                 EINSS7_I97TCAP_TERM_BASIC_END,
                                 appContextLength,
                                 appContext_p,
                                 0,     //userInfoLength,
                                 NULL); //userInfo);
      if(result != 0)
      {
         printf("Failed EndReq %d\n", result);
         Running = 0;
      }
   }while(result != 0);

    debug_printf(("EndReq sent, dialogueId: %u\n", dialogueId_b));
}
```

```
        else if(dialogueId_b > 0)
        {
           do
           {
              result = EINSS7_I97TBeginReq(Ssn,
                                           UserId,
                                           tcapInstanceId,
                                           dialogueId_b,
                                           priOrder,
                                           qualityOfService,
                                           destAdrLength,
                                           &destAdr[0],
                                           orgAdrLength,
                                           &myNodeAdr[0],
                                           appContextLength,
                                           appContext_p,
                                           0,       //userInfoLength,
                                           NULL);   //userInfo);

              if(result != 0)
              {
                 printf("Failed BeginReq %d\n", result);
                 Running = 0;
              }

           }while(result != 0);

          debug_printf(("BeginReq sent, dialogueId: %u\n", dialogueId_b));
        }
    }

    else
    {
        noOfParallelDialogues--;  /* Count down active dialogues */
    }

    if((Ssn == 19 || Ssn == 9) && (parameters_p_b[2] > 0))
    {
        XFree(UserId, newOperationCode_p);
    }

    return 0;
}

USHORT_T EINSS7_I97TEndInd(
    UCHAR_T           localSsn_a,
    USHORT_T          userid_a,
    EINSS7INSTANCE_T  tcapInstanceId_a,
    USHORT_T          dialogueId_a,
    UCHAR_T           priOrder_a,
    UCHAR_T           qos_a,
    UCHAR_T           compPresent_a,
    UCHAR_T           appContextLength_a,
    UCHAR_T          *appContext_p_a,
    USHORT_T          userInfoLength_a,
    UCHAR_T          *userInfo_p_a)
{
    debug_printf(("\n\tReceived: T_END_ind\n"));
    debug_printf(("\tDialogue ID = %d\n", dialogueId_a));
    (void)localSsn_a;
    (void)userid_a;
    (void)tcapInstanceId_a;
    (void)dialogueId_a;
    (void)priOrder_a;
    (void)qos_a;
    (void)compPresent_a;
    (void)appContextLength_a;
    (void)appContext_p_a;
    (void)userInfoLength_a;
    (void)userInfo_p_a;
    return 0;
}
```

```
USHORT_T EINSS7_I97TAddressInd(UCHAR_T ssn_l,
                               USHORT_T userId_l,
                               EINSS7INSTANCE_T tcapInstanceId_l,
                               USHORT_T dialogueId_l,
                               UCHAR_T bitMask_l,
                               UCHAR_T addressLength_l,
                               UCHAR_T *orgAdr_p_l)
{
    printf("ssn_l %c\n", ssn_l);
    printf("userId_l %d\n", userId_l);
    printf("tcapInstanceId_l %d\n", tcapInstanceId_l);
    printf("dialogueId_l %d\n", dialogueId_l);
    printf("bitMask_l %c\n", bitMask_l);
    printf("addressLength_l %c\n", addressLength_l);
    printf("*orgAdr_p_l %c\n", *orgAdr_p_l);

    printf("Received: T_ADDRESS_ind\n");
    return 0;
}

int main(int argc, char *argv[])
{
    LibraryInitializer init;
    INT_T going = 1;

    // Initiate Key & IV
    std::string passphrase = PASSPHRASE;

    S2K* s2k = get_s2k("PBKDF2(SHA-1)");      // string-2-key
    s2k->set_iterations(NOOFITERATIONS);
    SecureVector<byte> key_and_IV = s2k->derive_key(32, passphrase).bits_of();
    SymmetricKey newKey(key_and_IV, SIZEOF_KEY_AND_IV);
    InitializationVector newIv(key_and_IV + SIZEOF_KEY_AND_IV, SIZEOF_KEY_AND_IV);

    key = &newKey;
    iv = &newIv;

    if( argc > 1 && argc < 3 )
    {
        initAppId(argv);

        setUp(UserId);

        signal(SIGTERM, signalHandler);
        signal(SIGINT, signalHandler);

        TC2();
        sleep(2);
        TC1();

        if(*argv[1] == '1')
        {
            printf("main:\tI have SSN %d\n", Ssn);
            while (going)
            {
                chooseTestCase(&going);
            }
        }
        else
        {
            printf("main:\tSsn %d == Listener\n", Ssn);
            TC4();
        }
    }
    else
    {
        printf("Not the right amount of parameters\n");
    }
    tearDown(UserId);
    return 0;
}
```