NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MULTILEVEL SECURE FRONT END FOR DATA COMMUNICATIONS

by

Philip J. Corbett
March 1986

Thesis Advisor: Uno R. Kodres

CURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
. DECLASSIFICATION/DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)

. NAME OF PERFORMING ORGANIZATION aval Postgraduate School	6b. OFFICE SYMBOL (If applicable) Code 62	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
. ADDRESS (City, State, and ZIP Code) onterey, California 93943-5000		7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

. ADDRESS (City, State, and ZIP Code)

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO.

TITLE (Include Security Classification)

ULTILEVEL SECURE FRONT END FOR DATA COMMUNICATIONS

PERSONAL AUTHOR(S)
orbett, Philip, J.

3a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
aster's Thesis	FROM ______ TO ______	1986 March	113

6 SUPPLEMENTARY NOTATION

COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Multilevel Security, Information Security, Trusted Computer System, Communication Security, Gemini Computers

9 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis demonstrates the feasibility of using a multilevel secure computer system to augment traditional security measures used to safeguard sensitive information in an office to office communication environment. A multilevel secure communication interface can be used for high speed transmission of a wide variety of computerized information, from text files, to large volumes of bulk data including computer program listings. Such a system significantly reduces the delays associated with traditional transmission techniques such as couriers, and registered mail. The ability to encrypt all external communications provides additional security. By automating message processing functions, providing secure storage devices, and restricting access to sensitive information, the multilevel secure communication interface can greatly improve overall system security.

0 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED

2a NAME OF RESPONSIBLE INDIVIDUAL Uno R. Kodres	22b TELEPHONE (Include Area Code) 408-646-2197	22c OFFICE SYMBOL Code 52Kr

D FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete — SECURITY CLASSIFICATION OF THIS PAGE

Multilevel Secure Front End for Data Communications

by

Philip J. Corbett
Lieutenant, United States Navy
B.S., U. S. Naval Academy, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1984

ABSTRACT

This thesis demonstrates the feasibility of using a multilevel secure computer system to augment traditional security measures used to safeguard sensitive information in an office to office communication environment. A multilevel secure communication interface can be used for high speed transmission of a wide variety of computerized information, from text files, to large volumes of bulk data including computer program listings. Such a system significantly reduces the delays associated with traditional transmission techniques such as couriers, and registered mail. The ability to encrypt all external communications provides additional security. By automating message processing functions, providing secure storage devices, and restricting access to sensitive information, the multilevel secure communication interface can greatly improve overall system security.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in the research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

Some terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each occurance of a trademark, all trademarks appearing in this thesis will be listed below, following the firm holding the trademark:

1. Gemini Computers Inc., Monterey, California
   Gemini Trusted Multiple Microcomputer Base
   GEMSOS
2. Digital Research, Pacific Grove, California
   Pascal MT+
   CP/M-86
3. INTEL Corporation, Santa Clara, California
   INTEL
   Multibus
   APX-286

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. PROBLEM STATEMENT

This thesis investigates the use of a multilevel secure computer system as a secure front end for data communications in a small scale network environment. The specific application of the proposed system would be to protect incoming and outgoing messages from unauthorized access, as well as to ensure secure internal routing of classified information.

An example of the type of environment in which the system would be utilized is shown in Figure 1.1 . This structure is similar to that found in many project offices within the Department of Defense. Communications within a project cover a wide range of classifications, and include both military and civilian installations. Recent highly publicized security violations have underlined the need to make sure that communications channels both internal and external are properly protected. Currently, this protection is provided by a variety of physical and electronic means. Among these techniques are:

1) hardware encryption devices
2) secure teletype
3) secure radio communications
4) couriers
5) message scramblers
6) secure modems

Each method has specific strengths and weaknesses that can be exploited by a potential adversary. By far the most difficult problems with existing methods are to control access to the physical devices, and monitor internal distribution of received information. Access control is the responsibility of the security manager, however even in a

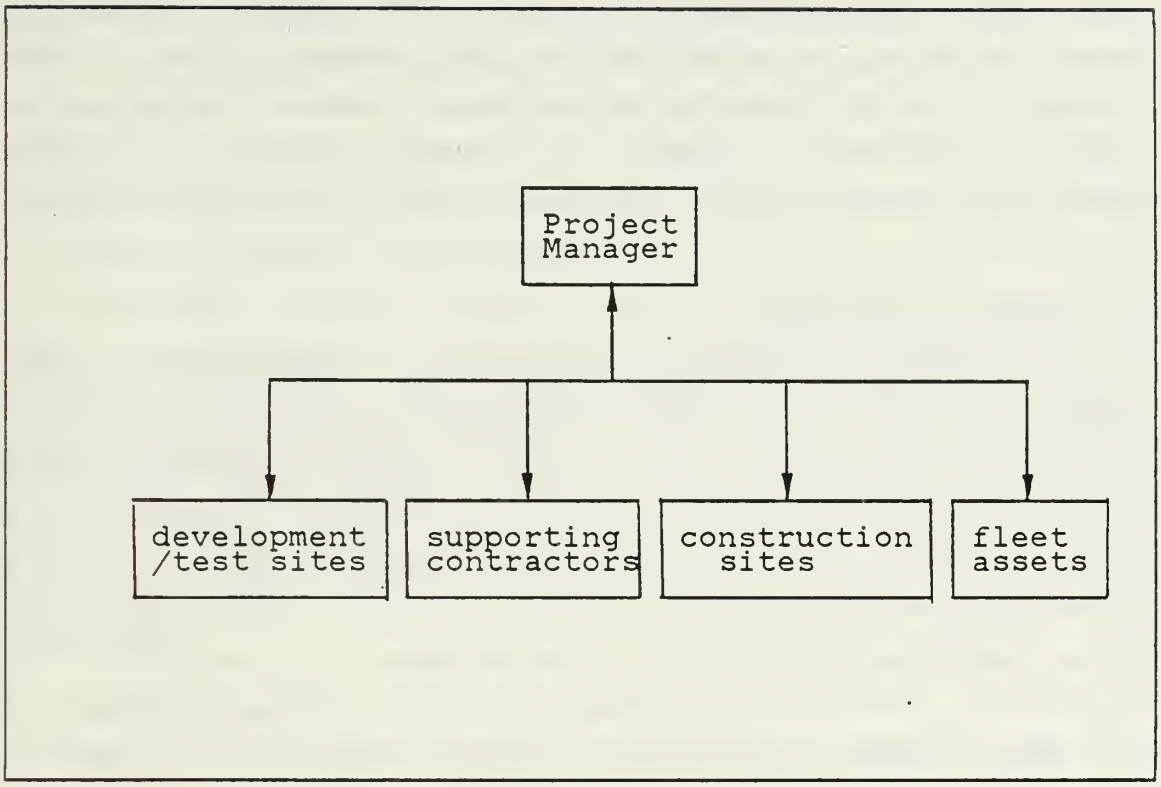


Figure 1.1 Sample Project Office Organization.

small project the problem can become unmanageable. This is especially true due to the recent proliferation of computerized processing systems throughout the Department of Defense (DOD) and commercial industry.

This research was performed in conjunction with the Naval Postgraduate School's AEGIS Modeling Group. This group is sponsored by the AEGIS Combat System Project Office to conduct research in the area of combat system development. The AEGIS project is made up of many field activities which include both military and civilian personnel. All activities receive, transmit, and are required to store classified information. Included in this information are messages, official correspondence, and computer programs related to AEGIS Combat System development. These documents are currently processed by the traditional methods discussed previously. The process is slow, and often reduces the amount of time an activity has to respond to an urgent problem. In addition to the external delays, once a document is received, it must go through internal security processing before it is delivered to the ultimate destination.

B. PROPOSED SOLUTION

This thesis proposes inserting a multilevel secure computer system as the trusted project communications interface and traffic manager. The trusted computer system [Ref. 1] would receive all incoming traffic, determine its classification, and notify the destination that it has an incoming message. If the destination did not have sufficient clearance to display the message, it would not be delivered. When transmitting data, the system would ensure that the transmission device is of the appropriate classification, and that the data is properly encrypted. By automating message handling and record keeping functions associated with the transmission of classified data, the transmission delay can be significantly reduced. The use of a trusted computer system in this capacity would also allow greater flexibility in establishing security policies. Each classification level can be further broken into several smaller groups in which access is based on a user's 'need to know' information of a particular type. This technique would enhance overall security by further restricting access within each security level.

The Department of Defense is currently evaluating several systems for approval to operate in this capacity. The Gemini Trusted Multiple Computer Base is the trusted computer system used in this research. A model for a secure communication system was developed which allows single level remote terminal users located at different sites, to communicate through a multilevel communication process created by the Gemini system. The Gemini trusted computer system is still undergoing development which imposed some restrictions on the scope of the communications system which was developed. These restrictions did not however, prevent demonstrating the feasibility of using a trusted computer system in this application environment.

Although primary concern is in protecting classified data, interception of large quantities of unclassified (for official use only) data can also be damaging. Documents which are by themselves unclassified, can be analyzed along with other intercepted information to produce a classified result. For this reason, all external communications throughout the model secure communication system are encrypted.

C. THESIS FORMAT

This thesis is composed of five chapters which are designed to provide the reader background information concerning multilevel security concepts, and then discuss the design of a model for the type of secure communication system discussed above.

Chapter I provides introductory information concerning the problem addressed in this research as well as the proposed solution.

Chapter II contains a discussion of multilevel security concepts. It explains the various types of security, and discusses the current Department of Defense (DOD) requirements for each type. General security methods are presented as well as methods used to attack secure systems. Data encryption methods are discussed, and a strategy for providing maximum data protection using the Gemini system's data encryption device is developed. The remainder of the chapter is devoted to explain Gemini system architecture, and discuss how it creates a multilevel secure environment.

Chapter III discusses actual Gemini system operation. The design of a model secure communication system is presented with a discussion of system constraints imposed by hardware and software limitations.

Chapter IV discusses system implementation and testing. Test results are used to demonstrate the system's ability to act as a secure front end for data communications between remote data terminals.

Chapter V brings together system test results to make a series of observations concerning the feasibility of utilizing a trusted computer system, such as the Gemini, as a multilevel secure front end for data communications.

II. BACKGROUND

A. MULTILEVEL SECURE COMPUTING SYSTEMS

1. Trusted Computer System Requirements

There are many documents which attempt to lay down requirements for trusted computer systems. They have been generated at all levels of the government, and in some cases are in conflict with each other. In 1983 an attempt was made within the Department of Defense (DOD) to bring together these documents as well as other information concerning trusted computer systems. The goal was to come up with a single source document which would define guidelines which could be used to develop and test new systems. The document which resulted from this research is entitled the "DOD Trusted Computer System Evaluation Criteria," more commonly referred to as the 'Orange Book' [Ref. 1]. Published in 1983, it contains definitions and information essential to understanding trusted computer systems. The Orange Book goes into extensive detail concerning the implementation of automated data processing (ADP) security systems. This thesis will primarily be concerned with the major issues involved in using a trusted computer system, and will not deal with actual implementation details. As described in [Ref. 1] there are two types of security policy to be considered. The first is mandatory security which is defined as:

"Security policies defined for systems that are used to process classified or other specifically categorized sensitive information must include provisions for the enforcement of mandatory access control rules. That is, they must include a set of rules for controlling access based directly on a comparison of the individual's clearance or authorization for the information and the classification or sensitivity designation of the information being sought, and indirectly on considerations of physical and other environmental factors of control. The mandatory access control rules must accurately reflect the laws, regulations, and general policies from which they are derived." [Ref. 1: p. 72]

As the name implies, mandatory security policy is a a strict limitation of access based on access level which is determined by the user's security clearance. This policy can not be changed and represents the foundation for the second type of security policy. Discretionary security policy is a subset of mandatory security policy which represents a further restriction of access to information based on a user's 'need-to-know' the information. The control objective for discretionary security is:

"Security policies that are defined for systems that are used to process classified or other sensitive information must include provisions for the enforcement of discretionary access control rules. That is, they must include a consistent set of rules for controlling and limiting access based on identified individuals who have been determined to have a need-to-know for the information." [Ref. 1: p. 73]

This type of security is a definite asset in a research and development environment. In particular, when developing combat system software, a project manager may have teams developing several modules simultaneously on the same system. Although the modules may be of the same classification level, the manager may want to limit each team's access to the module on which they are working. This would be accomplished by establishing a discretionary security policy.

Traditional attacks on security systems have involved compromise of keywords which would allow unauthorized access to a system. This threat can largely be eliminated by physical means: changing keywords, multilevel identification, and restricting access to the system. A more subtle attack, and potentially more dangerous threat is the establishment of a covert channel in the system. A covert channel is defined as "any communications channel that can be exploited by a process to transfer information in a manner that violates the system security policy." [Ref. 1: p. 79] In a multilevel computer system the presence of a covert channel can be exploited to gain unauthorized access to information without alerting security mechanisms. Covert channels will be discussed further in Chapter III as a design consideration for the multilevel secure communications system.

One of the most difficult tasks in developing trusted computer systems is determining test criteria to evaluate their performance. As the security level is increased, the test criteria become more stringent and detailed. When operating in a network environment, the problem is made even more difficult by requiring communications security between the trusted computer systems as well. This thesis is primarily concerned with this portion of the security problem. The Department of Defense is in the process of preparing a document which will detail evaluation criteria for trusted computer networks [Ref. 2].

2. Secure Communication Methods

As described by Voydock and Kent in [Ref. 3], there are two basic types of communications security. These are link-oriented and end to end security measures. Selection of a type of security measure for a particular application is dependent on the complexity of the network, as well as the vulnerability of the system to attack.

Link oriented measures treat each link in the communications chain from source to destination as a separate security problem. Each node is responsible for encrypting information passing through it, and for transmitting the information on the appropriate link. Encrypting with a different key at each node provides added security in that, compromise of one link does not necessarily mean that other links will also be compromised. This type of system does have several serious drawbacks [Ref. 4: p. 144]. First, in order to encrypt using a different keyword at each node means maintaining a large keylist. Changing keywords can be very costly. Second, since each link is encrypted independently there must be physical security at each node. Finally, in addition to physical security at the nodes, hardware and software components must be certified to process the security level of information passing through the node.

The second type of security measure is end to end protection. End to end security treats the network as a secure medium in which protocol data units (PDUs) are transported [Ref. 4: p. 145]. Since each link is not encrypted independently, interception of the message stream at an intermediate node will not necessarily compromise the information. In addition to being a great deal less expensive to implement, end to end encryption has several other advantages [Ref. 4: p.145]. Because there is no additional encryption at intermediate nodes there is no need for physical security at the nodes. Users or host computer systems can independently decide whether or not to use the security measures, further reducing the cost. Finally, end to end encryption can be used in both packet switched and packet broadcast network environments, whereas link oriented security measures are more difficult to adapt in a packet broadcast system [Ref. 5: p. 213].

Figure 2.1 is a simplified diagram of the type of communications system this thesis is proposing. The system is relatively small scale with a limited number of users.

The communication network consists of modem-like telecommunications. Link oriented security measures are much too complex for this type of application. They would also provide no significant advantage over end to end measures. End to end security measures were chosen for this design to ease implementation and trouble-shooting. In this application the host computers are assumed to be trusted computer systems. The host is being used in a secure front end configuration therefore the end to end security measures will only be used to connect the hosts. This simplifies the problem by limiting the number of hardware and software interfaces involved in the end to end encryption path.

3. Network Security Threats

Before developing a trusted computer network, it is necessary to understand how an intruder could try to exploit system weaknesses. Voydock and Kent [Ref. 4] divide the methods of attack into three categories. These categories are:

1) unauthorized release of information
2) unauthorized modification of information
3) unauthorized denial of use of resources

The first type of attack is passive while the second and third require active involvement by a potential intruder. In passive attacks an intruder places himself in a communication path and monitors traffic flowing over the links. Even with the information encrypted the intruder can still gain knowledge about the types of information being transmitted, and the destinations to which it is sent. By examining the message length and transmission frequency the intruder gains additional information. One form of this attack uses a 'Trojan horse' program to establish a covert channel and alter message characteristics which would passively divert copies of information to the intruder [Ref. 6].

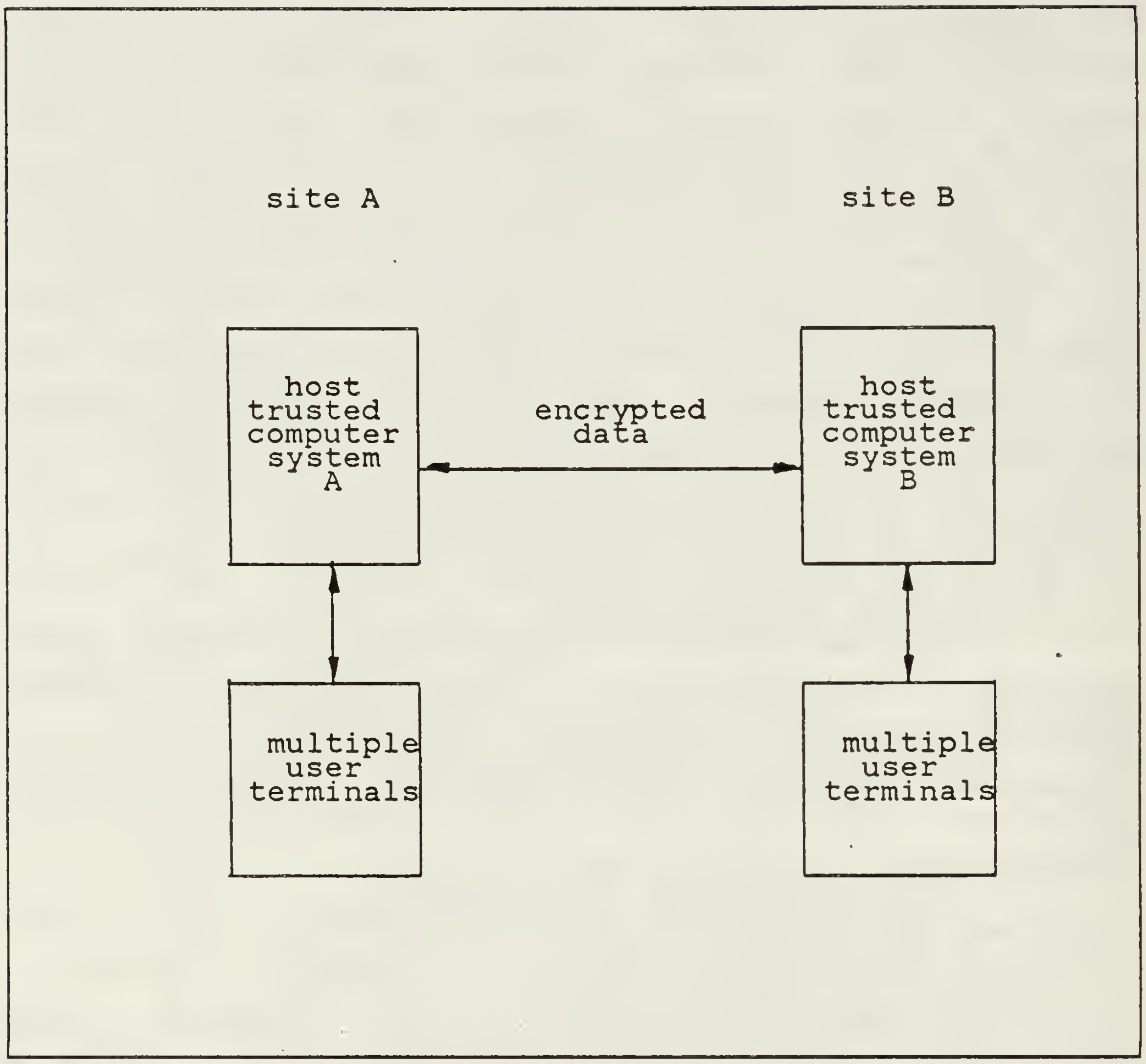


Figure 2.1 Simplified System Design.

Active attacks involve more risk to the intruder, however, they can yield much more damaging results. These attacks are normally directed at the protocol data units (PDUs). Once access is gained to the PDU chain the stream is modified in a manner dependent on the objective of the intruder. The category of active attacks can further be subdivided into three basic techniques [Ref. 7]:

1) message stream modification
2) denial of message service
3) spurious association initiation

Message stream modification attacks seek to alter the authenticity, integrity, and/or ordering of the PDUs [Ref. 4: p.142]. In attacking authenticity the source or destination of a PDU is altered causing information to be misdirected. This is similar to the passive attack. The intent is to disrupt communication more than to passively obtain information. Attacks on message integrity involve the data portion of the PDU. Modifying or deleting information can cause transmitted data to be misrepresented. Finally, changing the order of the PDUs can make the message unintelligible to the user trying to receive it.

The second type of active attack, denial of message service, can take two forms. The first type is complete denial in which a communications channel is blocked allowing no PDUs to pass. In the second form all PDUs are delayed making it impossible to decode the incoming message. These attacks are difficult to detect, particularly if they are put into effect between messages so that the user has no indication that communications have been interrupted.

Spurious association initiation, the third type of attack, is a form of jamming. In this attack a previous recording of communications between two authorized users is played back to confuse the receiver into thinking it is receiving legitimate PDUs.

After examining the methods of attacking secure networks, a plan to counter these threats needs to be developed. Voydock and Kent [Ref. 3] point out that there are limitations on the ability to detect and prevent these types of attacks. They say that "Although message stream modification, denial of service, and spurious association initiation attacks can not be prevented, they can be reliably detected. Conversely, release of message contents and traffic analysis attacks usually can not be detected but they can be effectively prevented." Given these limitations, they present five goals for providing communications security: [Ref. 4: p. 143]

1) prevention of release of message contents
2) prevention of traffic analysis
3) detection of message stream modification
4) detection of denial of message service
5) detection of spurious association initiation

Referring to Figure 2.1 it can be seen that in the system proposed by this thesis, there are two general areas in which an attack could occur. The first is within the host computer system. In this application the host is the Gemini Trusted Multiple Microcomputer Base. The second possible area is the communications network itself. Communications on these links need to be encrypted in a manner that will provide the maximum possible protection for the encryption method chosen. The remainder of this chapter will discuss how data encryption, and Gemini system features can be used to achieve the desired security goals.

4. Data Encryption

Data encryption is fundamental to a secure communications network. The methods available vary widely as do the security levels for which they are approved. Approval is based on the computational power, and the amount of time required to break the code. A cipher that cannot be proven to resist all attacks is considered 'computationally secure' if the computational cost involved in breaking it exceeds the value of the information gained [Ref. 8]. Recent technological advances have produced computer chips which reliably encrypt data with a high degree of security. The relatively low cost and high speed of these devices make them excellent choices for secure network applications. The major problem to date has been getting them approved for transport of DOD classified data. Two major encryption methods are the Data Encryption Standard (DES) [Ref. 9], and the Public Key systems [Ref. 8]. The Gemini system used in this research utilizes DES as it's encryption method, and therefore it will be the only method discussed.

The Data Encryption Standard (DES) is the National Bureau of Standards (NBS) cryptographic protection standard [Ref. 10]. It is widely used for the protection of commercial data. It has come under attack from several sources [Ref. 10: p. 171]. Because of these alleged weaknesses DES is not currently authorized for transmission of DOD classified data. Despite its problems DES remains a highly secure and reliable method of encryption for official documentation which would otherwise be transmitted in unencrypted form. As discussed in Chapter I, interception of large volumes of unclassified data can often lead to unintended compromise of classified information. The remainder of this section will discuss characteristics of DES encryption and techniques which can be used to maximize the protection of transmitted data.

There are four modes that the DES can operate in. They are: the Electronic Code Book (ECB) mode, the Cipher Block Chaining (CBC) mode, the Cipher Feedback (CFB) mode, and the Output Feedback (OFB) mode [Ref. 11].

a. Electronic Code Book (ECB) Mode

Figure 2.2 shows how a DES device operates in this mode. ECB is the simplest of the DES modes however, it is also the most vulnerable to attack. This is because identical blocks of cleartext code will always produce identical ciphertexts until the encryption key is changed. This method is not recommended for transmitting messages which contain repetition of data forms such as English text messages [Ref. 10: p.178]. Since identical blocks yield identical ciphertexts, by observing over a period of time an intruder would eventually be able to determine the cleartext message.

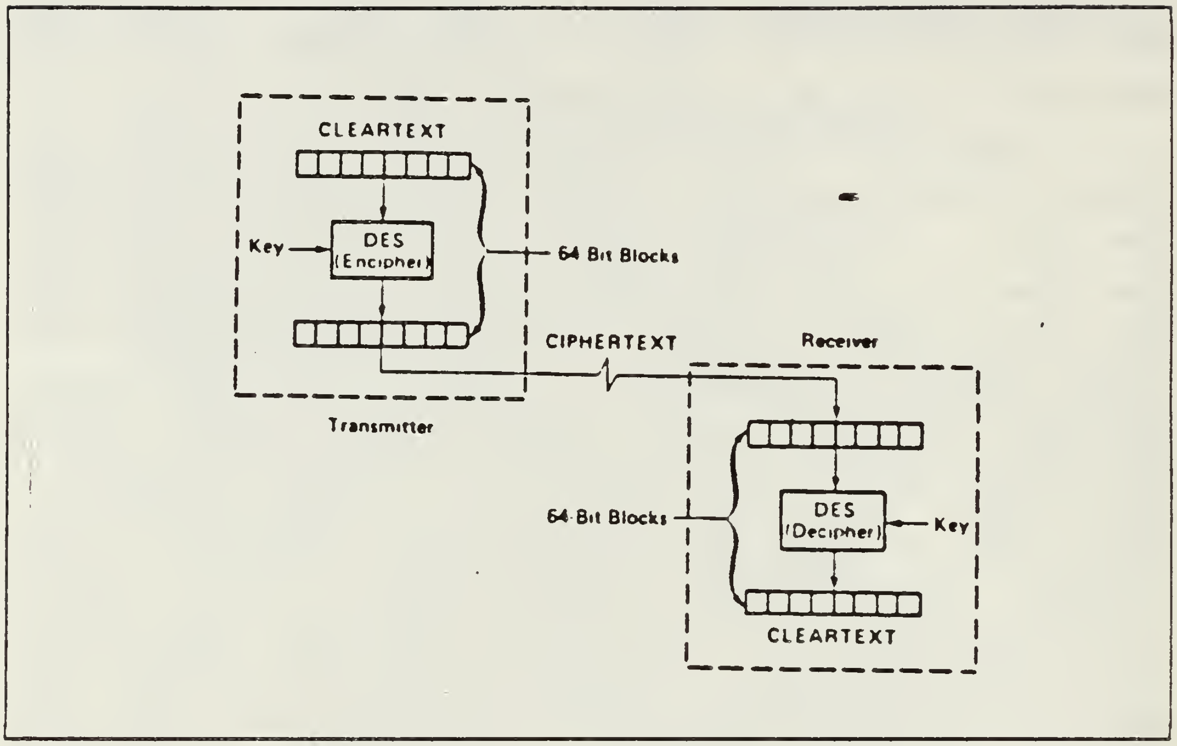


Figure 2.2 ECB mode of DES encryption.

b. Cipher Block Chaining (CBC) Mode

Figure 2.3 shows how the CBC mode operates. CBC is a block encryption method which overcomes the pattern recognition problems of ECB mode by using the ciphertext of each preceding block as an input to encrypt the next block. The process is started by applying an initialization vector to the first block of data to be encrypted. Incomplete blocks are padded as additional protection against pattern recognition attacks.

c. Cipher Feedback(CFB) Mode

Figure 2.4 shows the CFB mode of operation. CFB mode is a stream encryption technique in which a key stream is generated, then combined with plain text to produce a ciphertext. The ciphertext is then fed back as an input to the key stream generation process. Stream ciphers are in general slower than block ciphers [Ref. 4: p. 151], and are not used when large throughputs are required.

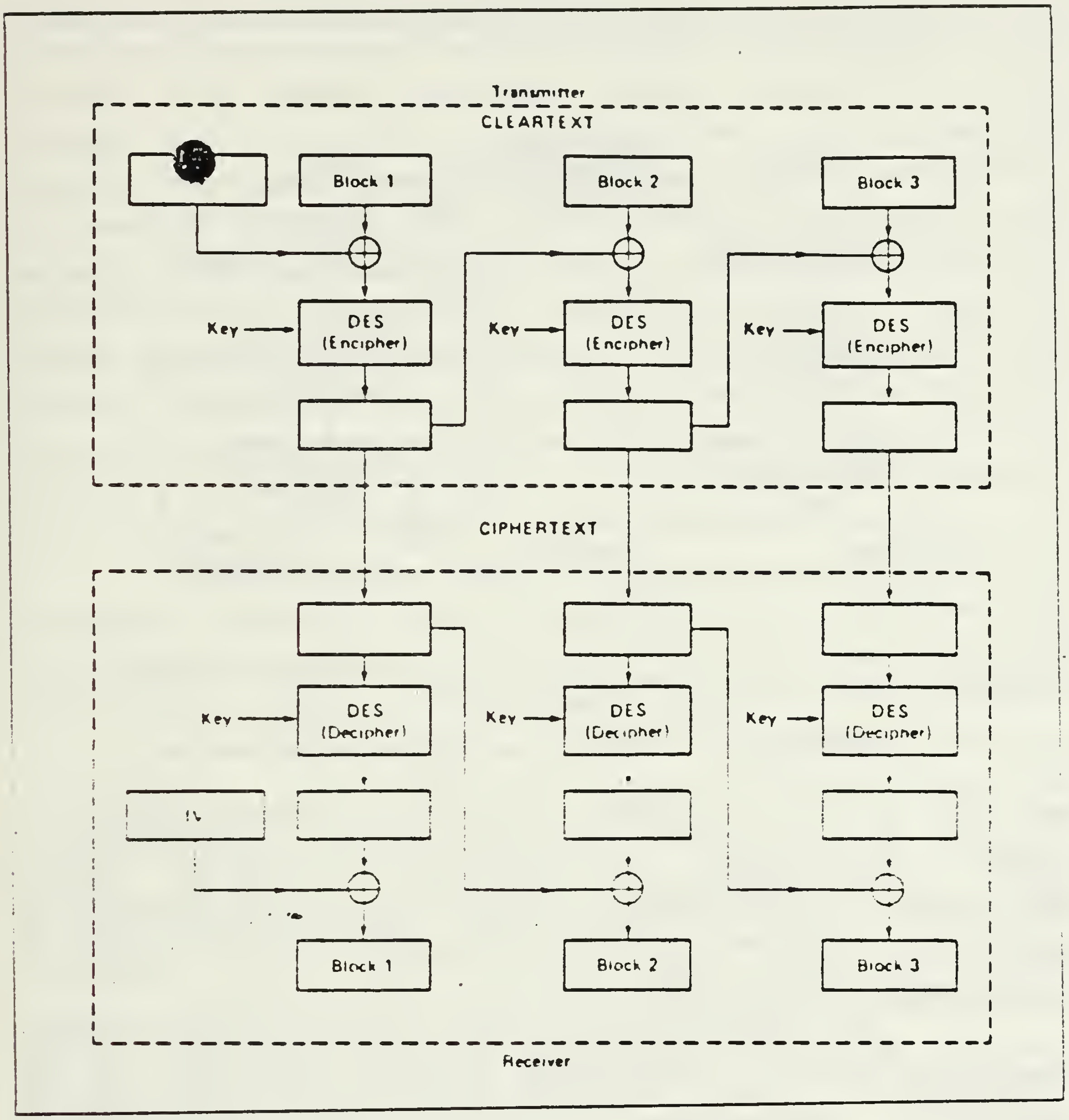


Figure 2.3 CBC mode of DES encryption.

d. Output Feedback (OFB) Mode

The OFB mode is also a stream encryption method. In this method the key stream is completely independent of the plaintext and ciphertext streams. This eliminates the problem of error propagation and would seem to be a definite advantage. However, some degree of error propagation is

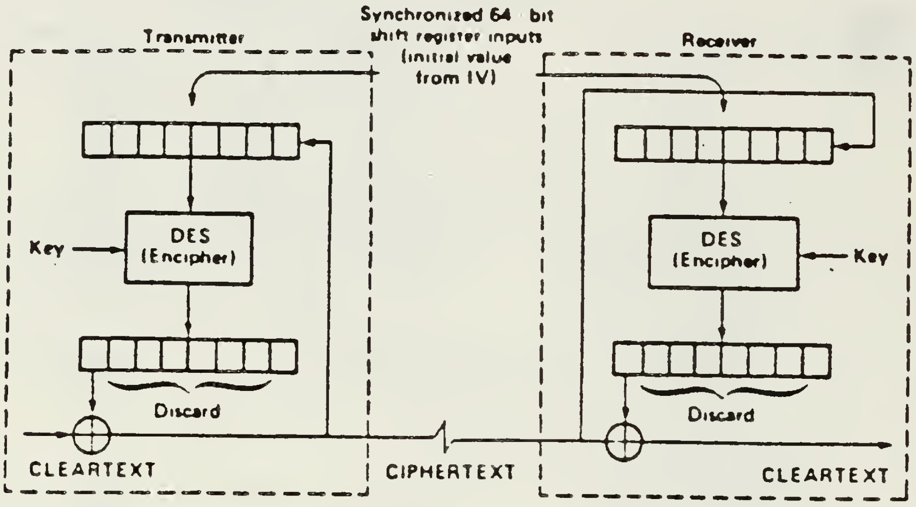


Figure 2.4 CFB mode of DES encryption.

required to detect message modification attacks [Ref. 4: p.149]. As a result, OFB mode is not normally used in secure network environments. This mode is not implemented on the Gemini system's hardware encryption device because it is not self synchronizing.

The communication system being developed in this thesis can best be implemented using the CBC mode of DES encryption. As discussed in Chapter I the system must be capable of quickly handling large volumes of data (large throughput), as well as official correspondance. Specific steps can be taken to strengthen the CBC mode against the types of attack presented earlier in this chapter.

The first method of attack was to force unauthorized release of message contents. There are several ways to prevent release of message contents using CBC mode encryption. Control of encryption keys and their use are very important in preventing attacks of this type. One technique is to encrypt the PDU contents using one key, and encrypt the network protocol address information using another [Ref. 4: p. 153]. This provides a sort of 'two man control' over the transmitted message. To prevent pattern recognition attacks, the operator must ensure that each message starts with a unique prefix. Since each ciphertext depends on the encryption of the previous block, this will ensure that each message produces a unique ciphertext. This can be accomplished by employing a communication protocol which generates a unique message identifier, or by changing the CBC initialization vector for each message and transmitting it with the message.

The second method of attack is through traffic analysis. End to end security measures are more susceptible to traffic analysis attack than link oriented measures [Ref. 4: p. 157]. As discussed earlier in this chapter, link oriented security measures were not feasible for this application. As a result, the task will be to minimize the susceptibility of the end to end system to this type of attack. Figure 2.5 shows the ISO reference model of open system interconnection. Voydock and Kent [Ref. 3] show that encryption below the transport layer does not provide significant additional cryptographic protection. Encryption at this level also provides the maximum reasonable degree of protection against traffic analysis attacks. By encrypting the source and destination information the intruder is limited in his analysis to the host computer level. Even then, the attacker can only examine the quantity, frequency, and lengths between the hosts while protecting the identity of the source and destination of the information.

Countermeasures used to detect message stream modification attacks are related to the communication protocol employed by the system. A wide variety of

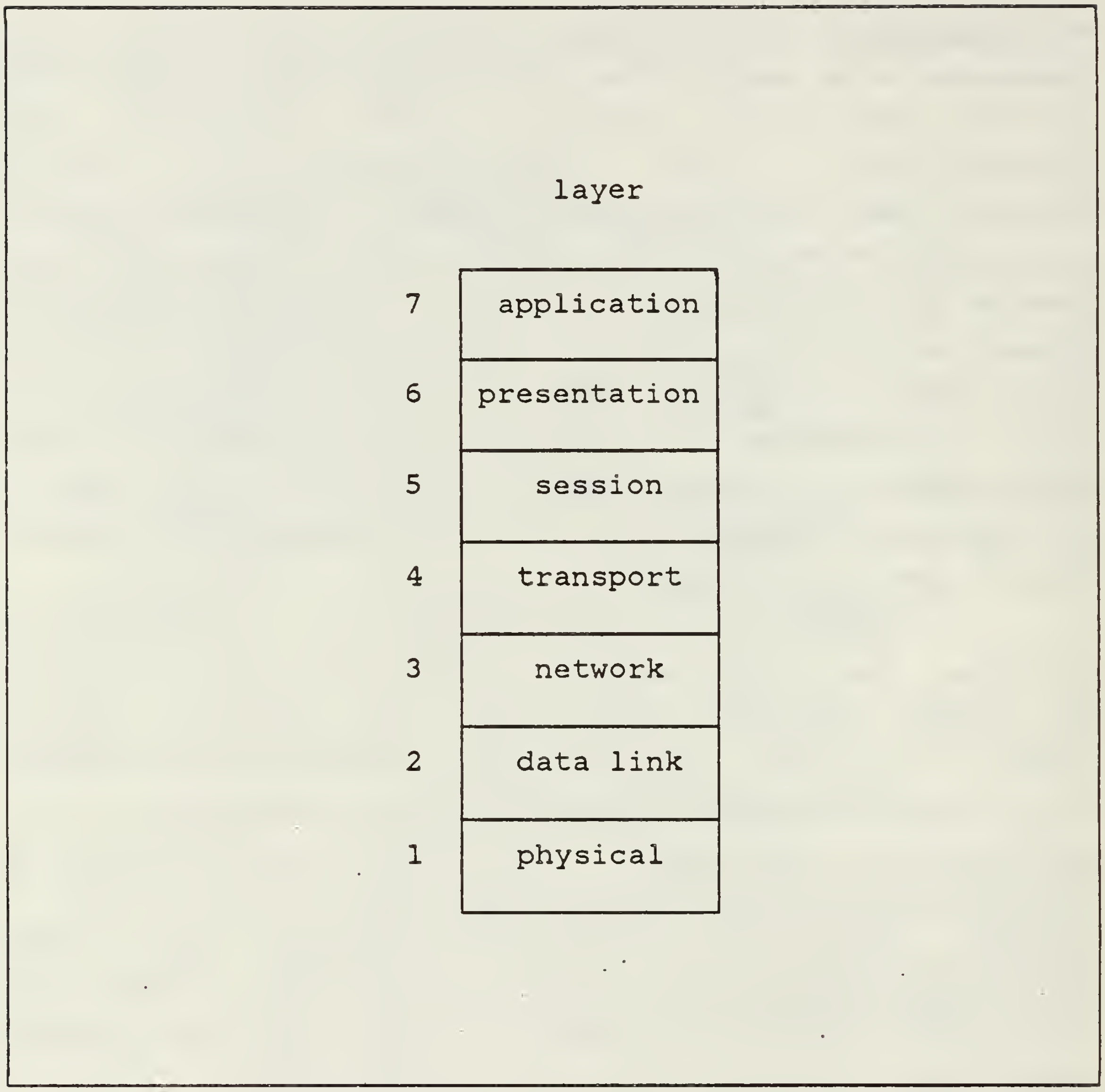


Figure 2.5 ISO Interconnection Reference Model.

protocols are currently in use throughout DOD and commercial industry. Because any secure communications network will most likely adapt an existing protocol, no optimal protocol will be proposed. The system will make use of existing Gemini system features to ensure message authenticity and integrity. If a communications protocol was found to have insufficient protection against message stream modification attack, it could be then be strengthened by adding additional transmission verification features to the transport layer.

Detection of denial of message service attacks involves verifying that the communications channel between the two hosts is open. This is best accomplished in an encryption environment by exchanging request-response PDUs [Ref. 4: p. 165] at random intervals. Failure to respond to this PDU indicates that a denial of service attack may be in progress. This technique obviously slows down the system. By selecting an appropriate frequency for the checks based on the types of messages being exchanged, the effects of this slow down can be minimized.

Spurious association initiation attacks can be detected using the same method used to counter denial of message service attacks. By sending the request-response PDUs at random intervals, 'play-back' attacks can be reliably detected. Another method, using periodic intervals, would be to send a time verification in the request-response PDU.

5. Summary

The secure communications network proposed in this thesis has two major areas of vulnerability: the host computer system (Gemini system), and the communications network. End to end security measures were chosen between host computer systems because of the relatively small size of the network, and for ease of implementation. DES encryption was selected for network encryption because it is widely used, and is readily available as the Gemini system's data encryption device. Although this method is not authorized for transmission of classified data it could be combined with another approved encryption method to transport classified information. This technique is called layered encryption [Ref. 12], and will be further discussed in Chapter III. The CBC mode of DES encryption was selected to best meet the needs of this application. The communications network can further be strengthened against attack by taking the following steps:

1) Change encryption keys as often as is feasible taking into account the expense involved, and the threat environment in which the system operates.
2) Encrypt data PDUs and address information with separate keys if possible.
3) Ensure each message starts with a unique message identifier to hamper pattern recognition attacks.
4) Encrypt data in the transport layer to provide maximum cryptologic protection.
5) Use request-response PDUs, exchanged at random intervals to verify that communications channels are open.

These features have been incorporated in system design to the maximum degree possible. The next section will discuss the Gemini Trusted Multiple Microcomputer System, and how it provides security at the host computer level.

B. GEMINI TRUSTED MULTIPLE MICROCOMPUTER BASE

1. Description of Gemini System Components

The Gemini Trusted Multiple Microcomputer Base is one of many systems currently being evaluated by the Department of Defense Computer Security Center for certification to operate at the B3 [Ref. 1] level of classification. Until recently lack of evaluation criteria, as well as microprocessor technology made construction of such systems impractical. The foundation on which all trusted computer systems are developed is the security kernel. While operation of the security kernel will be discussed in general terms, details concerning kernel construction are beyond the scope of this research. The Gemini system employs the latest technology in both hardware and software engineering. Some of it's major features are [Ref. 13]:

1) The capability to operate up to eight Intel APX-286 microcomputers in parallel. This provides tremendous processing power, while communicating through shared memory increases throughput.

2) The Gemini system is extremely flexible with regard to the types of peripheral devices which may be connected to the Multibus. These include fixed hard disk, removable disk, and high density floppy diskette drives, as well as non-volatile memory devices. A maximum of eight devices may be attached to each RS-232 I/O interface board.

3) With its multiple microcomputers, the Gemini system supports a variety of multiprocessing and multiprogramming applications. Processes can be pipelined to a single processor, or distributed in parallel among several processors.

4) Other features include a NBS DES chip encryption device, real time clock, and non-volatile memory to protect passwords and encryption keys.

The Naval Postgraduate School (NPS) version of the Gemini system is a subset of the full delivery system. This system has one APX-286 microcomputer, (2) 1.2 Mbyte floppy disk drives, and one RS-232 interface board (max. 8 ports). The NPS system does have other specific limitations which will be discussed in Chapter III.

The Gemini system also provides a self-hosting environment for software development [Ref. 13: p. 4]. This allows users to develop applications software. The original intent of this thesis was to generate application software using the Janus/Ada computer language. The Janus/Ada environment was not available in time to support the research, and as a result Pascal MT+ was used instead. Pascal MT+ programs must be modified to run with the Gemini Secure Operating System (GEMSOS). Not all Pascal MT+ constructs are supported in the GEMSOS environment. The majority of modifications occur in the input/output area. Because communications to and from devices require special formats, a utility library is provided with the system containing routines which put calling arguments in the proper format for use in GEMSOS. These special features will be discussed in more detail later in this chapter.

A major source of attraction for the Gemini system is its tremendous potential for future growth. Its ability to handle a variety of hardware configurations is especially valuable in DOD applications where a trusted computer system may be required to communicate with systems using different protocols and hardware interfaces. When utilized as proposed by this thesis as a secure front-end for data communications, the Gemini system could potentially communicate simultaneously with a variety of secure communication devices using different I/O ports.

2. Gemini Resource Management Overview

The Gemini Secure Operating System (GEMSOS) kernel is logically divided into three management areas. These are: segment management, process management, and device management. Management functions are invoked by initiating a GEMSOS service call [Ref. 13: p. 5]. The formats for calling arguments are found in the GEMSOS interface libraries provided with the Pascal MT+ compiler.

a. Segment Management

All data utilized in the GEMSOS environment is contained in segments. The applications programmer is mainly concerned with code segments, stack segments, and data segments. Bootstrap and kernel segments normally do not change when developing basic applications software on the NPS system. There are eight segment management functions. A discussion of how to initiate these service calls is contained in [Ref. 14: pp. 13-78]. Segments can also be managed in groups. Secondary storage devices are represented by volumes which can be identified as separate entities to a calling process. Volumes and individual segments can be brought into the address space of the calling process by using resource management service calls.

b. Process Management

The NPS Gemini system currently has only one processor, however through process management functions it is able to support a full range of multiprogramming and multiprocessing applications. Each process is identified by code, stack, and data segments which uniquely identify the process. Once created the process can be synchronized to run simultaneously with other processes using one of two methods. Eventcounts and sequencers were selected over other possible techniques because they are particularly well adapted to operation in a secure environment [Ref. 13: p. 6]. All segments created in an applications program are assigned an eventcount and sequencer automatically. Process management calls to these devices allow the programmer to coordinate process functioning while maintaining access security.

c. Device Management

The third management area is device management. The Gemini approach to device management is to minimize the size of the security kernel code by reassigning device management functions to application level code whenever possible [Ref. 13: p. 8]. This has two effects. Reducing the size of the kernel makes verification easier, however it also makes writing I/O applications software more difficult. Traditional input and output files are replaced by segments which can be read from or written into. Devices are attached and detached to allow them to be used by more than one calling process. Process synchronization primitives are used to control access to the segments made available to an attached device. The I/O device controller is treated as a process, which is then synchronized with the available segments eventcounts or sequencers to perform the required device management functions. Additional information concerning Gemini resource management functions is contained in [Ref. 13: pp. 5-11].

3. Gemini Secure Operating System(GEMSOS) Architecture

The Gemini system uses four hierarchical rings to implement its security structure. Ring 0 provides the most security, while ring 3 is least secure. It can support both discretionary and nondiscretionary policies. The nondiscretionary or mandatory policy is controlled in ring 0. This policy cannot be modified. Ring 1 is used to control the discretionary or 'need to know' policy, supervise the use of the data encryption device, and support any other security functions not contained in the mandatory policy. Rings 2 and 3 are available to the programmer for use in developing applications software.

The security mechanism which coordinates inter-ring communications involves the control of access to subjects and objects. A subject is a process which is allowed to operate over a specific domain within the system. An object is a specific piece of information which is assigned a security label. All access between subjects and objects is controlled by the GEMSOS security kernel located in ring 0. Approval is based on a comparison of the security labels of the two entities trying to gain access.

Security labels are used to identify the access class of all subjects and objects. The access class is further broken into a compromise (observe) level, and an integrity (modify) level. Compromise and integrity protection are based on strict properties which must be observed in order for access to be granted. Figure 2.6 is taken from [Ref. 13: pp. 16,17], and contains a simplified statement of these properties. Domination as stated in these properties means that the level of the access component is greater than or equal to the entity it is trying to observe or modify.

Compromise protection property 1 is a traditional security policy. It states that in order to observe information, you must have a clearance equal to or greater than the information you want to observe. The second property is more subtle. This property prevents, for example, a secret user from modifying a file which could then be observed by a confidential user. This property is especially important in

Compromise Properties:

1) If a subject has "observe" access to an object, the compromise access component of the subject must dominate the compromise access component of the object.

2) If a subject has "modify" access to an object, the compromise access component of the object must dominate the compromise access component of the subject.

Integrity Properties:

1) If a subject has "modify" access to an object then the integrity access component of the subject dominates the integrity access component of the object.

2) If a subject has "observe" access to an object then the integrity access component of the object dominates the integrity access component of the subject.

Figure 2.6 Compromise and Integrity Properties.

prevention of 'Trojan horse' type attacks [Ref. 6]. The integrity protection properties are similar to the compromise properties except that they refer to the ability to modify information. Property 1 states that in order to modify a confidential document you must have at least a confidential integrity level. The second integrity property prevents, for example, secret users from observing (and possibly being influenced by) information which could be modified by someone with a lower integrity level.

In addition to the access class integrity described above, the Gemini system also employs ring integrity. Ring integrity means that subjects at a certain level can only access objects of the same, or a higher ring number. This policy reinforces the hierarchical structure of the GEMSOS rings.

These compromise and integrity properties are further complicated by the fact that the Gemini system is a multilevel system. This means that both users and resources may have clearance to access a range of security levels. Multilevel subjects are potentially very dangerous because within their range of operation they are not subject to the second compromise and integrity protection properties [Ref. 13: p. 20]. It is up to the applications programmer to ensure that he does not create subjects which will allow violation of these properties. This is especially important when interfacing with devices. Figure 2.7 is taken from [Ref. 13: pp. 21,22], and represents a summary of the security properties of single and multilevel devices.

Device access levels refer to the physical security of the environment in which the device is going to operate. This is separate from the security level of the process which is attempting to communicate using the device. For example, a terminal located in an unsecure room with an unclassified device access level, cannot receive information from a secret process. The term single level device implies that the maximum and minimum access classes for the device are the same. In multilevel devices they are different, and represent the range over which the device is allowed to operate.

4. Summary

The Gemini Trusted Multiple Microcomputer Base is an extremely capable computer system which combines state of the art technology with a high degree of

Single level devices-

1) To receive ("read") information:
process maximum compromise >= device minimum compromise
device maximum integrity >= process minimum integrity

2) To send ("write") information:
device maximum compromise >= process minimum compromise
process minimum integrity >= device minimum integrity

Multilevel devices-

1) To receive ("read") information:
process maximum compromise >= device maximum compromise
device minimum integrity >= process minimum integrity

2) To send ("write") information:
device minimum compromise >= process minimum compromise
process maximum integrity >= device maximum integrity

Figure 2.7   Single and Multilevel Device Properties.

flexibility to be able to handle a variety of possible applications. Its multiple processor and multiprogramming configurations are valuable assets when functioning as a secure front-end for data communications as proposed by this thesis. By being able to simultaneously handle devices with different protocol requirements and security levels, the Gemini system operating in this mode could potentially eliminate the need for separate stations for each secure communication device.

Key to developing a trusted computer system application is the ability to develop a sound, secure resource management strategy. This strategy must adhere to the system's mandatory security policy, and avoid misapplication of resource management functions which could make the system vulnerable to a covert channel attack.

Chapter II introduced trusted computer system concepts, and provided necessary background information concerning the Gemini system. Chapter III will discuss Gemini system operation, and the creation of an application program which will allow the system to function as a multi-level secure front-end for data communications.

III. SYSTEM DESIGN

A. DESIGN ISSUES

1. Objectives

The primary objective of this design was to develop a simple communications system which would demonstrate how the Gemini Trusted Multiple Microcomputer Base could be effectively utilized as a secure front end for data communications. There were three major phases in developing the system design:

1) Establish two way communications between users at remote terminals using the Gemini system as an external communications interface.

2) Use the Data ciphering Processor (DCP) [Ref. 14: p. 57] to provide end to end encryption of external communications.

3) Demonstrate the use of Gemini security mechanisms to prevent unauthorized access to classified information.

In order to create a realistic communications link it was necessary to simulate having two separate trusted computer systems communicating with each other. This was accomplished by having the Gemini system communicate with itself using separate I/O ports. By routing the incoming and outgoing traffic from each port to separate processes, the two computer environment was simulated. The system is operated by a system security manager who is located at a data terminal. The system security manager is responsible for:

1) System start-up and initialization.
2) Assigning access levels for user terminals.
3) Control of communications at the external ports.
4) Insertion of cryptographic keywords.
5) Routing of incoming traffic to the appropriate terminals.

Each user terminal is assumed to be located in an area which provides appropriate physical security for the access level of the terminal. Each can enter messages to be sent, transmit messages, and display incoming messages provided the terminal at which they are located has the proper access level.

To accomplish the second objective, the data ciphering processor (DCP) was used as discussed in Chapter II. The entire outgoing message, including address information, is encrypted to provide maximum protection. This technique is valid for this specific application because of the small size of the network, and the limited quantity of information being exchanged. It may not be appropriate in larger scale applications involving larger networks such as the Defense Data Network (DDN). Communications are assumed to be established between the two sites using some form of modem-like telecommunications device. In this design, the RS-232 external communications ports were directly connected by an interface cable. Since all incoming traffic must pass through the trusted computer first, it is up to the Gemini system to decipher the address information, determine the access class, and route the message to the proper incoming message buffer.

Additional security could be provided by encrypting the data a second time prior to transmission using another method. This technique is called layered encryption [Ref. 12: p. 159]. The second method could be an authorized DOD hardware encryption device, a secure teletype, a message scrambler, or an interface to a secure network such as the Defense Data Network (DDN). The Gemini system would format outgoing messages, and route them to the proper device.

The final goal, to test security of information and access, was demonstrated using a series of specific configurations and data sets to exercise security mechanisms. These tests are meant to demonstrate, rather than prove that information security and integrity are preserved. They are in no way intended to be exhaustive, however will allow for a series of observations to be made concerning overall system security.

2. Design Constraints

The hardware and software limitations of the Naval Postgraduate School (NPS) Gemini system limited the scope of system design. The NPS system has eight ports available for attachment of I/O devices. This would appear to allow for at least four user terminals in addition to the two communications ports and system security manager terminal. This is not the case due to a limitation on the number of process local device slots(8) which identify the I/O devices. The serial read and serial write devices must have separate process local device numbers assigned. Therefore two process local device numbers are required to attach a terminal as a read/write device. This situation is further complicated by the requirement that the read encryption, write encryption, read decryption, and write decryption devices must be attached separately also. Device management was a major factor in determining ultimate system configuration.

Software development constraints were generated by the environment in which this type of system would be utilized. An assumption was made that when acting as a secure front end, the system would most likely be adapted to an existing computerized processing system. There are a wide variety of such systems currently in use both within the Department of Defense (DOD) and commercial industry. Each system has specific built in physical and software security attributes. For this reason, no effort was made to provide security between the trusted computer system and the remote data terminals. These lines are assumed to be secure, as are the locations in which the terminals are utilized. In order to provide overall system security, these security measures would have to be verified prior to installation of the trusted computer system at a particular activity. Another assumption was that the system could potentially communicate by a variety of means including; secure teletype, secure landline, Autodin, or DDN. For this reason, no specific communications protocol was adopted. A source and destination header was placed at the start of each message along with initialization information for the data encryption device. This header could be further modified to allow the message to be transmitted over a particular communications network. The simplified source and destination header will be sufficient for purposes of this research.

3. Summary of Design Decisions

Figure 3.1 shows a block diagram of the final system design. Due to the process local device slot limitations discussed in the preceding section, only two user terminals were used. To provide additional flexibility, the access class of each terminal can be set and changed by the system security manager. All communications leaving the external communications ports are encrypted using the Data Encryption Standard (DES) operating in Cipher Block Chaining (CBC) mode. Communications between the trusted computer system and the remote data terminals are not encrypted, however are assumed to travel in a physically secure environment. System operation is controlled by the system security manager. User terminals can send messages to and receive messages from the trusted computer system, however they must rely on the system security manager to actually transmit the messages.

At first glance it appears to be a relatively simple task to create a single process which would allow messages to be exchanged between users. Figure 3.2 shows an example of how a process like this would operate, and why the design

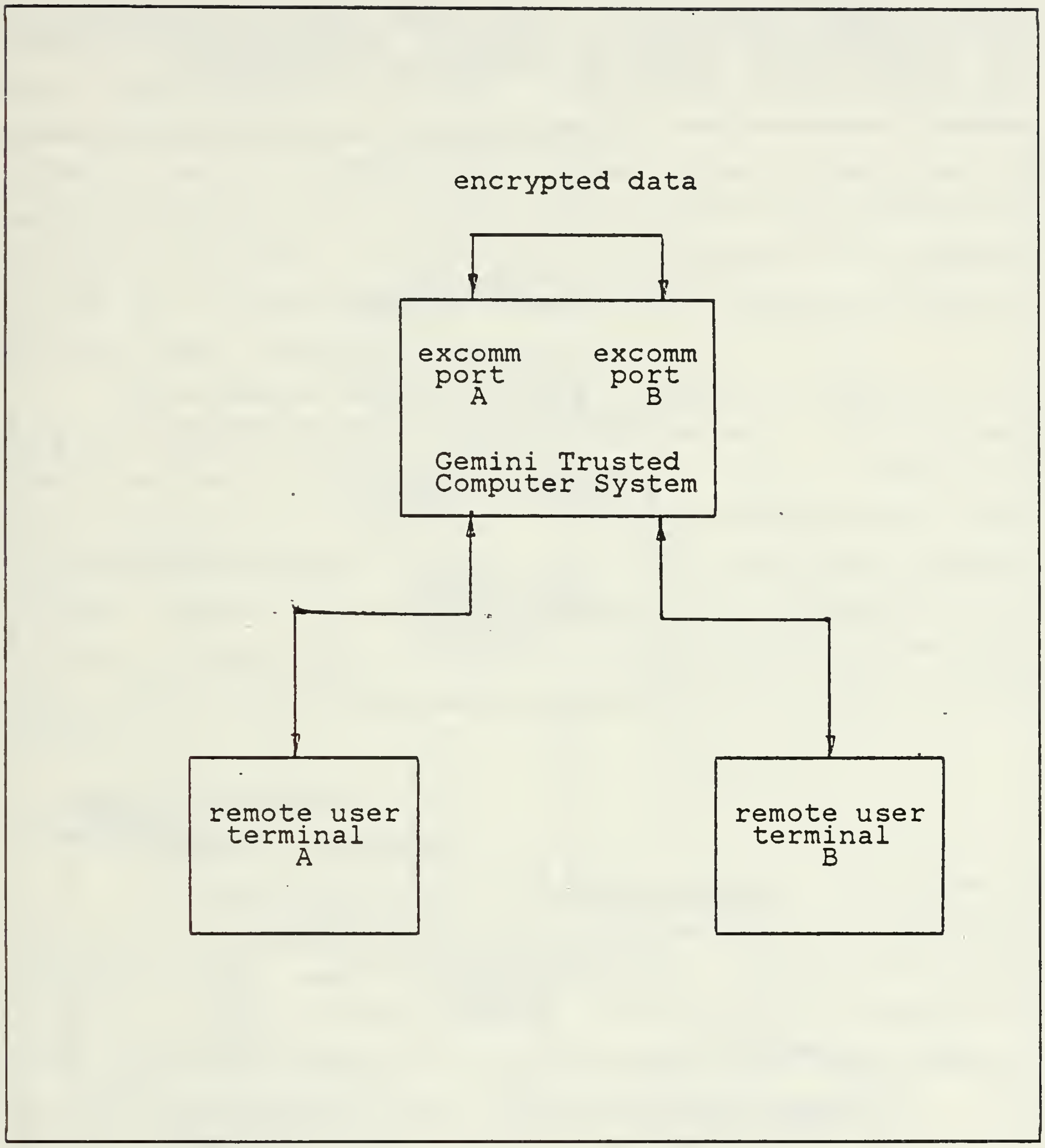


Figure 3.1 Final System Design.

would not work. The problem is caused by the multilevel nature of the communications process. In order to handle messages with different access classes, it must be a multi-level process. Attaching the single level terminals directly to a multilevel process creates the potential for a covert channel [Ref. 1: p. 79] which could be exploited to gain unauthorized access to classified information.

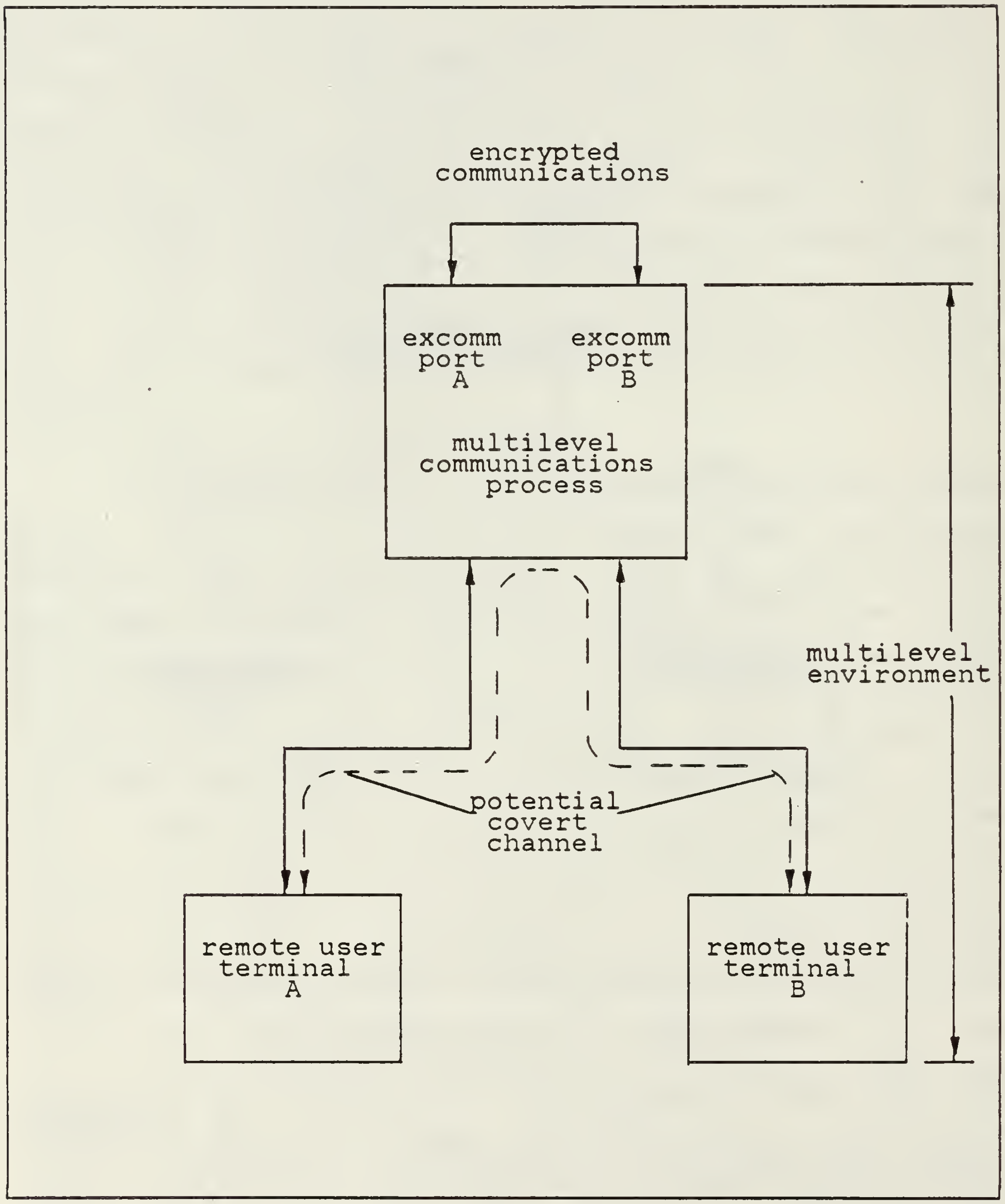


Figure 3.2 Process Block Diagram With Covert Channel.

To eliminate this problem, it is necessary for the system security manager to create a single level process for each user terminal attached. Figure 3.3 eliminates the covert channel problem by providing a single level process buffer to protect information. Even if an attacker was to cause information of a higher level to be misrouted by the multilevel process, it would still be protected from compromise by the single level process which interfaces directly with the user terminal.

This design creates another problem. That is, the need for synchronization among the processes. Interprocess communications are synchronized by using eventcounts [Ref. 15: p. 20]. Although the system simulates two separate trusted computer systems, only one multilevel communications process was used to simplify the synchronization problem. Since the communications processes would be identical this limitation did not adversely impact system design.

B. SYSTEM IMPLEMENTATION

1. Hardware Components

As discussed in the preceding section, the number of data terminals used in the system was limited by the number of process local device slots available. Figure 3.4 shows the final system hardware design. Terminal 0 is used by the system security manager to initiate and coordinate communications via the external communications interface. Two remote user terminals are also connected as read/write devices. They represent the users at the two sites which are exchanging information. The external communications interface consists of a special cable which allows one of the ports to function as a data communication equipment port (DCE) while the other functions as a data terminal equipment (DTE) port. All of the Gemini ports are initially configured as DCE ports. In order to communicate computer to

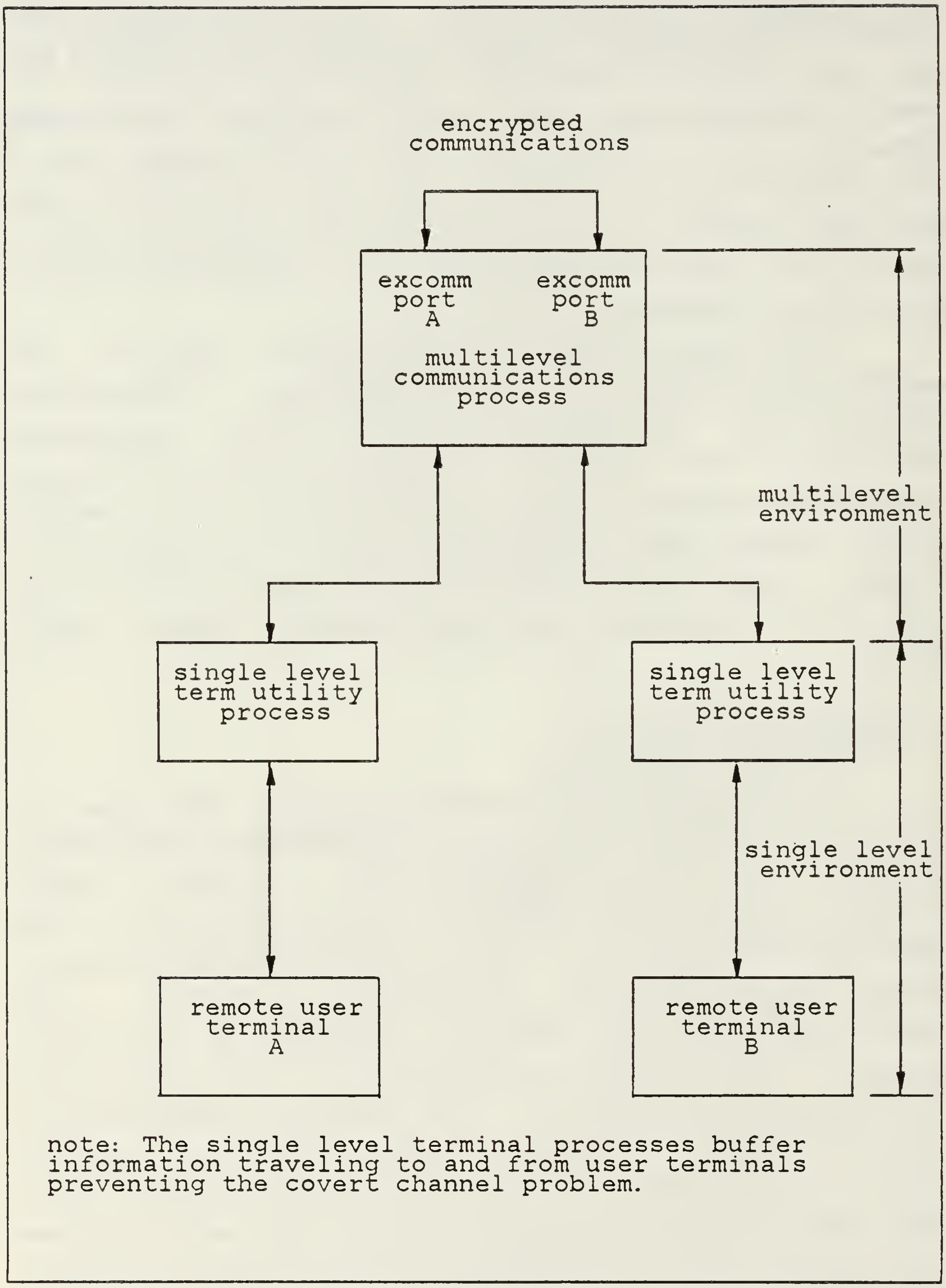
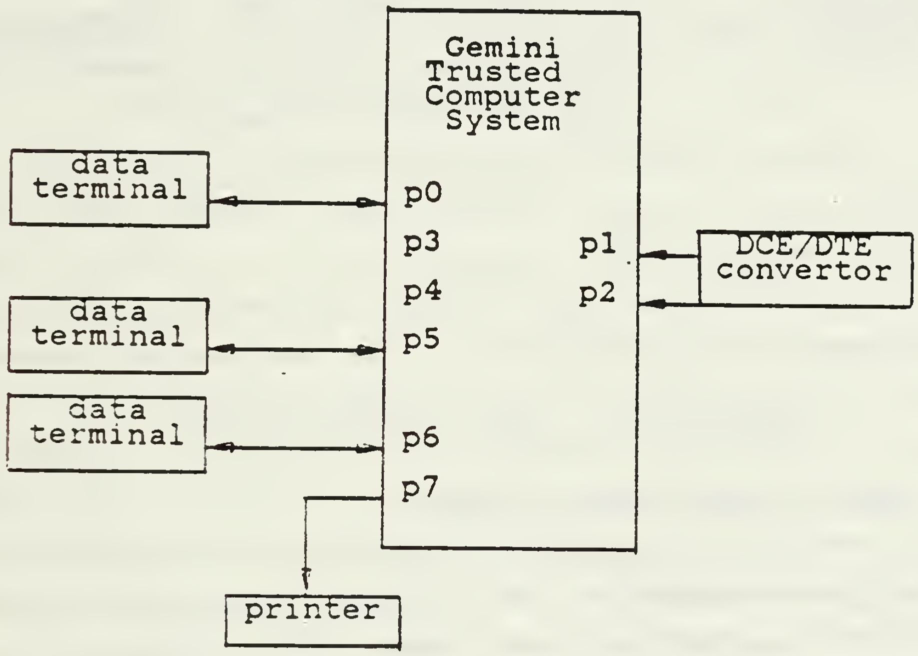

encrypted communications
excomm port A
excomm port B
multilevel communications process
multilevel environment
single level term utility process
single level term utility process
single level environment
remote user terminal A
remote user terminal B
note: The single level terminal processes buffer information traveling to and from user terminals preventing the covert channel problem.

Figure 3.3 Process Diagram Eliminating Covert Channels.

computer required the use of a special DCE to DTE convertor cable [Ref. 14: p. 51]. Figure 3.5 shows how the convertor cable is constructed. Gemini ports 3 and 4 were not used in this application.



port assignments:

p0- system manager terminal
p1- external communication port 1
p2- external communication port 2
p3- not used
p4- not used
p5- remote terminal user 1
p6- remote terminal user 2
p7- printer

Figure 3.4 Final Hardware Diagram.

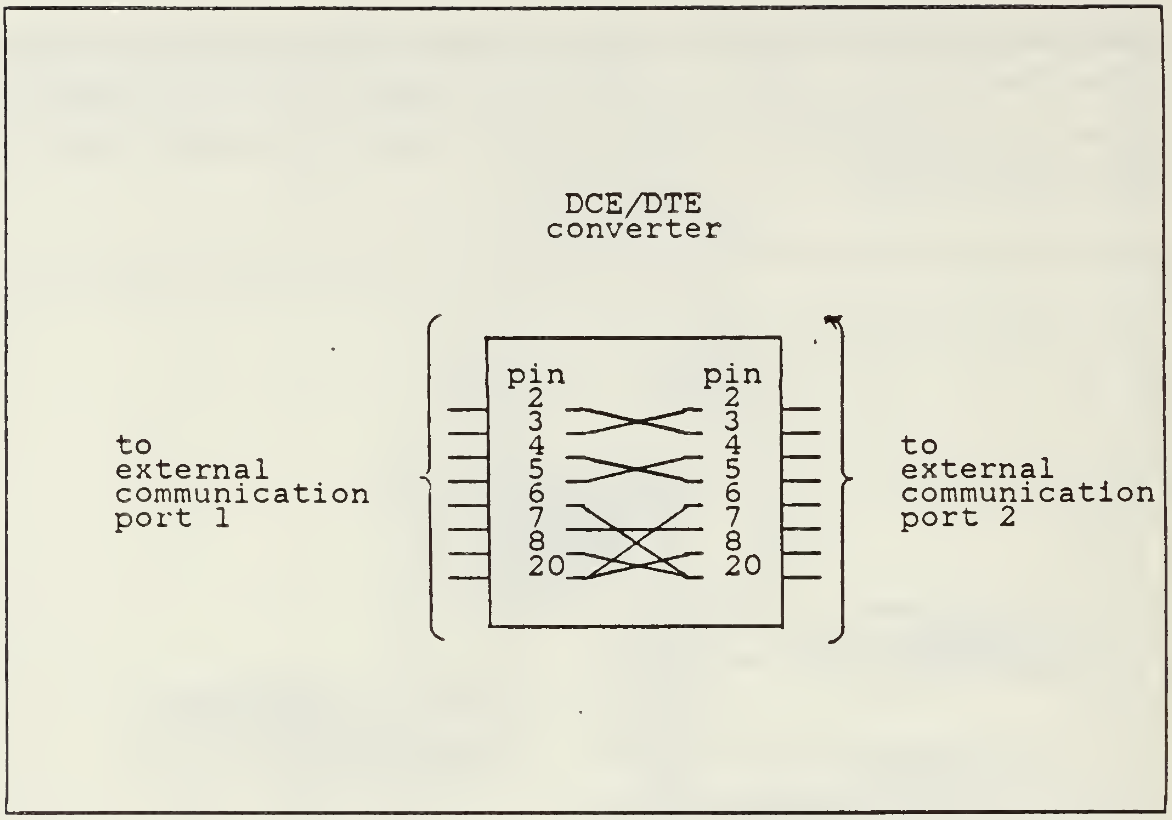

DCE/DTE converter
pin
pin
2
3
4
5
6
7
8
20
2
3
4
5
6
7
8
20
to external communication port 1
to external communication port 2

Figure 3.5 DCE to DTE Convertor.

2. Application Program Format

Preparing programs to run in the Gemini Secure Operating System (GEMSOS) environment is significantly more complicated than running the Pascal MT+ programs in a non-secure environment. In order to be accepted by the system they must first be put into a specific format which can be recognized by the Gemini Secure Operating System (GEMSOS), to gain access to the security kernel. There are several software tools which can greatly speed up the process of preparing a program to be run in the secure environment. The fact that a program compiles successfully does not necessarily mean that it will run in the GEMSOS environment. Following Pascal MT+ compilation, the program is linked to the appropriate modules using a file named, 'application_name.KMD' [Ref. 16]. This file contains a formatted list of the modules the application segment needs to be linked with. The result of the linking process is a file named 'application_name.CMD' which still has no security classification assigned. To assign security classification, and prepare the program to execute in the secure environment, a secure volume must be created by running the operating system generation (SYSGEN) program.

Executing the SYSGEN program includes the application program into a segment structure which is then transformed into a "bootable system segment structure on formatted volumes." [Ref. 17: p. 1] Detailed procedures for using the SYSGEN program are contained in [Ref. 17: pp. 8-18]. The key to proper use of the SYSGEN program is identifying the segment structure in which the application segment is going to be placed. The segment structure includes the boot-strap, kernel, application code, and data segments. The easiest way to identify this segment structure is to include it in a submit file named 'application_name.SSB.' For basic application programs, the segment structure does not change. Use of the submit (.SSB) file eliminates the need to enter the segment structure interactively each time the operating system generation program is run. Use of the SYSGEN submit mode is further explained in [Ref. 17: pp. 13-18].

C. SYSTEM SOFTWARE DESIGN

1. Application Segment Development

Application software for this system was developed using modular programming construction techniques. This allowed for independent testing of each module prior to its inclusion in the main program. This technique was especially useful because trouble-shooting GEMSOS related Ring 0 service calls was particularly difficult. Figure 3.3 shows the three processes which were developed as application code segments. They are:

1) multilevel system manager process

2) terminal A single level terminal utility process
3) terminal B single level terminal utility process

Each process was developed as an independent application code segment. The terminal utility segments are almost identical, however must remain separate entities because they represent different systems. In addition, they are assigned different physical ports and can also be assigned different access levels.

a. Terminal Utility Segments

As discussed earlier in this chapter the user terminals can input, transmit, and display messages. Each terminal is a single level device capable of sending and receiving messages of the same level. Figure 3.6 shows a flow diagram for the terminal utility application segment. The actual code for the Pascal MT+ program which implements this flow diagram is contained in Appendix A. This program is activated when the terminal process is created by the system manager process.

All messages input and received at the terminal are stored in a specially designated message buffer segment. Access to this segment is shared by the user terminal and the system manager process. Each terminal has its own message buffer segment, and cannot access the other's segment without going through the system manager process. When the user has completed his message transactions, he initiates a logoff procedure.

The logoff procedure deletes the terminal process and returns the resources allocated to the process to the system manager process which created it. These include memory space, process local segment numbers, and any attached devices.

b. System Manager Segment

The system manager segment controls system configuration, data encryption, and communications through

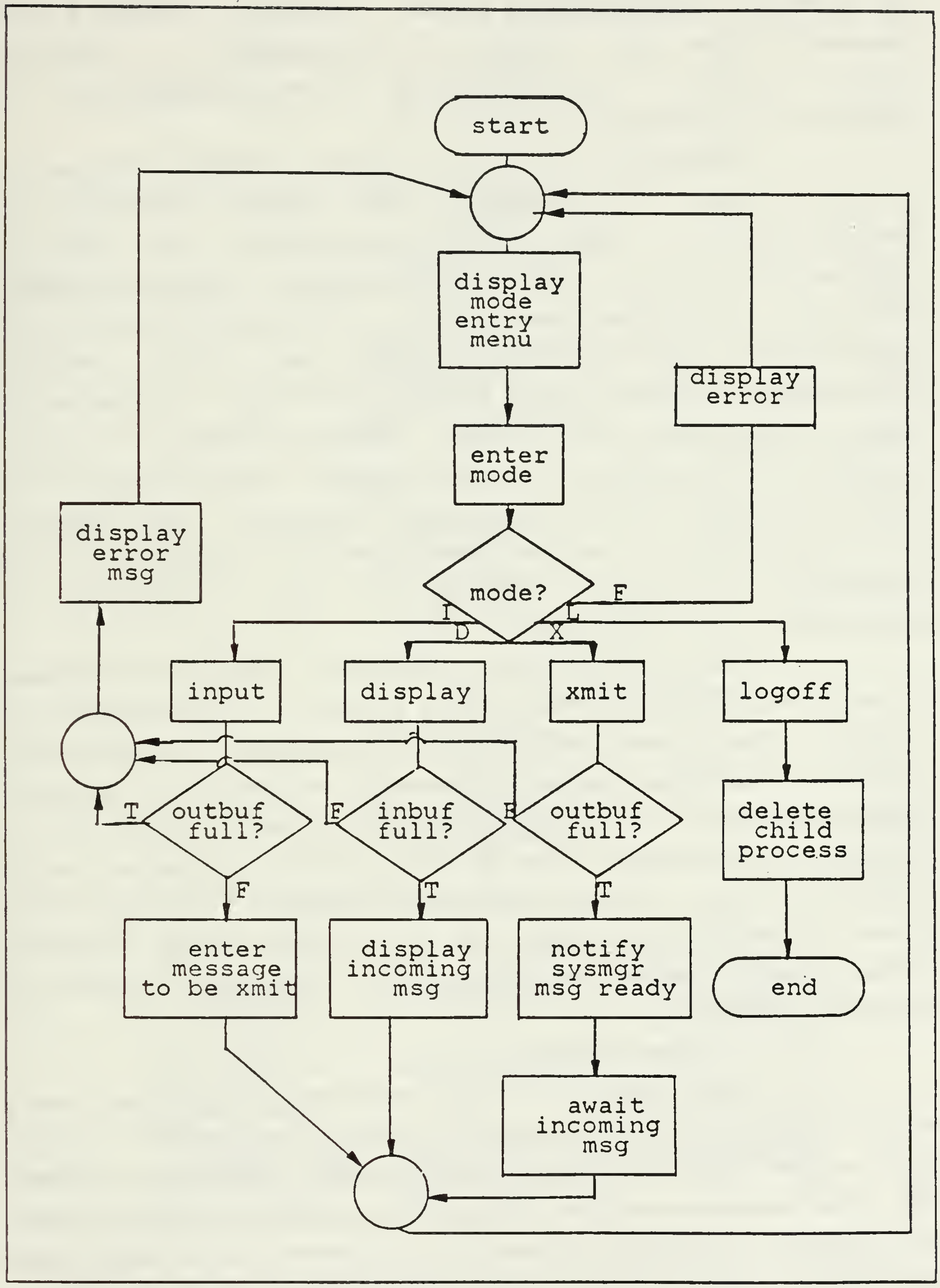


Figure 3.6 Terminal Utility Flow Diagram.

the external communications ports. Figure 3.7 shows a flow diagram of how this segment is constructed. A detailed source listing of the Pascal MT+ code implementation is contained in Appendix B.

Creation of a child process requires completion of four record structures. Each record structure has several entries. Each entry is completed in a specific order which builds to the 'create_process' resource management call. Detailed instructions for process creation and record entry format are found in [Ref. 14: p. 28]. Segment and process management are the most difficult concepts for someone unfamiliar with secure computer systems to grasp. The procedure developed in this segment could be used as a model for process creation in other programs. The specific entries may vary, however the physical structure of the procedure is general enough to fit a variety of applications.

The system security manager located at terminal 0 has direct control over system assets. To provide this control, the system manager has the option of specifying (within predefined limits) how the system will operate. These parameters are entered when the system is initialized. They are interactively entered into a system operator record from which they can be drawn when required by other procedures. Parameters which do not need to be directly controlled by the system manager are fixed and cannot be directly accessed.

c. Program Documentation

Each module in the application segments has a header describing its purpose and general operation. Since this is the first research effort using the Gemini system, the intent was to provide clear programs which could be used as a basis for future research. In some cases this meant sacrificing efficiency in order to provide better clarity.

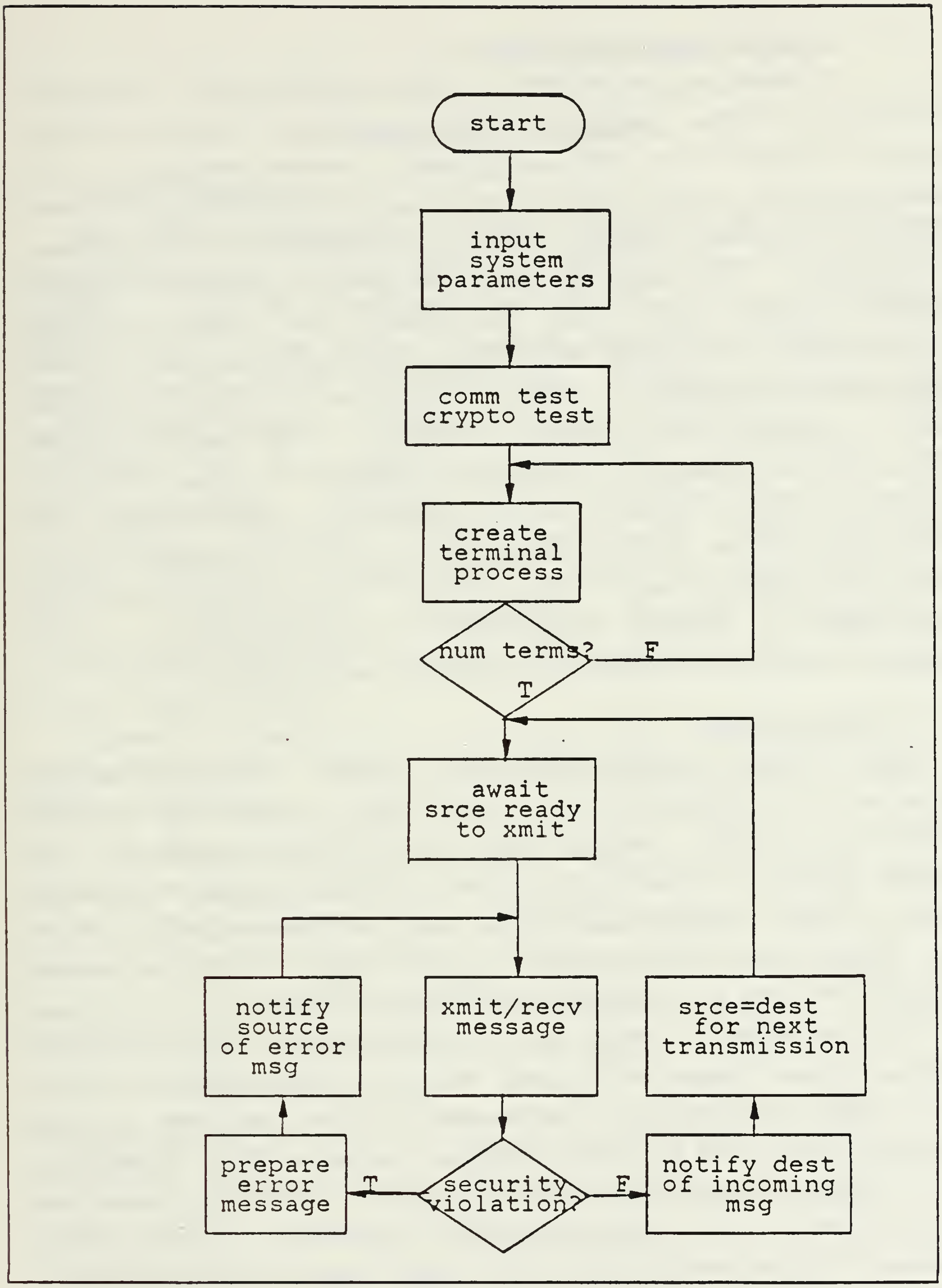


Figure 3.7 System Manager Flow Diagram.

2. Process Synchronization

Process synchronization was accomplished using the eventcount of the message buffer segments of each terminal process created by the multilevel system manager. By advancing the proper stack eventcount the terminal process alerts the system manager that it is ready to begin message processing. The terminal advances the the outgoing message buffer segment eventcount to notify the system manager process when it desires to transmit a message or when it has completed processing. When a terminal process indicates that it desires to transmit a message, the system manager transmits the message, and then unblocks the other terminal process to display the incoming message by advancing its incoming message buffer eventcount. This process can be continued indefinitely. The actual implementation of this sequencing is further explained in the applications code segment listings contained in Appendices A. and B.

D. DESIGN SUMMARY

This chapter has discussed the system design process in terms of its objectives and limitations. Hardware limitations of the NPS Gemini system limited the scope of the system design, but did not prevent achieving desired design goals. The resulting hardware and software configuration was implemented using modular construction techniques which greatly reduced the number of software errors.

The resulting system utilizes the Gemini as a two-way communications interface, and message processing facility. All communications are protected to the maximum extent possible using the Data Encryption Standard (DES) algorithm in the cipher block chaining (CBC) mode. Terminal processes are assigned single level access which eliminates the covert channel problem and prevents the user from gaining unauthorized access to classified information.

IV. DISCUSSION OF RESULTS

A. SYSTEM OPERATION

The model communication system developed in this thesis to demonstrate the feasibility of using a trusted computer system as a secure front end for data communications met or exceeded all design goals. Messages were passed between two remote terminals in a manner that ensured security from unauthorized access at both source and destination. Data encryption was utilized to maximize the security of the transmitted data. Finally, by varying the access class of the terminal processes it was possible to demonstrate the system's ability to detect and respond to security violations. Flexibility in determining system configuration allows modification of system parameters to meet a variety of test requirements.

System operation is initiated and controlled by the system manager. The multilevel system manager process creates the single level terminal processes at an access level predetermined by the system manager. Once initialized a remote terminal may only display and enter messages which are of the access level at which the terminal process was created. It is important to note that the user does not assign the classification of the outgoing message. Message classification is assigned by the system manager according to the access level of the terminal sending the message. This is done to prevent a user from downgrading a classified message to send to an unclassified user at another terminal. All security checks are therefore performed within the system manager process. For test purposes, the terminal access levels in this system are manually entered by the system manager. If the project manager did not want to leave this choice up to the system manager, the access level information could be hidden in a file that he does not have access too. Once it is started, the system operates independently. This eliminates the possibility of a corrupt system manager from manually misrouting information stored in the message buffers.

One potential problem was the possibility that an unclassified user could enter classified information in an unclassified message and transmit it to an accomplice who had tapped into the external communications line. To help prevent this, the outgoing message is encrypted using keys which are inserted by the system manager. Possible compromise of the key could further be prevented by having the key entered by someone other than the system manager. The goal of this process was to develop a system in which no one person would be in possesion of enough information to misroute, and potentially compromise classified information. There are a wide variety of possible system configurations. Selection of a particular configuration would have to be based on a detailed study of the activity, and its associated security requirements.

B. SYSTEM TESTING

1. General Comments

The process of debugging and running applications programs proved to be much more time consuming than had been originally anticipated. Three factors contributed to this problem. They were:

1) unfamiliarity with multilevel security concepts
2) difficulty in transforming Pascal MT+ programs to code compatible with the trusted computer operating system
3) time delays required to prepare modified programs to be tested in the secure environment.

As with any new area of study, multilevel security has its own terms and concepts which must be thoroughly understood prior to attempting to use the trusted computer system. As discussed in Chapter II, the manner in which the

Gemini system manages resources is very different from traditional non-secure systems. The interaction of the process, segment, and device management functions is key to understanding overall system operation.

The second problem concerned identifying Pascal MT+ instructions which were not recognized by the trusted computer operating system. A program which compiles without error, may not necessarily run in the secure environment. An example of this would be a program which contains the Pascal command 'read' or 'write'. These are legitimate commands which would compile without error. The problem arises when the secure operating system encounters the command. Trying to read or write to a file is not allowed in the Gemini Secure Operating System (GEMSOS). The file in this case would have to be redefined as a segment to which the process has access. The data would then be passed to and from the segment by using a pointer to the desired location. The best way to overcome this problem is to start with very simple programs which test specific functions and gradually build to larger more capable programs.

The final difficulty had to do with the amount of time that was required to take a program which had been compiled and prepare it to be tested in the secure environment. As discussed in Chapter III, in order to prepare a program to run in the secure environment, a secure volume containing the program segment must be created by running the operating system generation (SYSGEN) program. Once the secure volume is created, the system is reinitialized using the secure application program volume. When a problem is encountered in the execution of the program, the system will either execute an interrupt trap halt and indicate the processor's register contents at the time of the interrupt, or in some cases will halt completely. In either case, the error must be corrected before the system will be able to progress any further in the program. Once the desired correction has been made, the preparation process must be repeated to test the modified program. For the programs developed in this application, the preparation process took from between four and seven minutes for each program. The use of modular programming techniques is vital when programming in this environment to minimize the time delays associated with program execution and testing.

As future versions of the Gemini system become available, it is expected that the effects of these problems will be significantly reduced. Expanded system libraries, and an improved application development environment will make the process of writing programs which can be run in the GEMSOS environment simpler and less time consuming.

2. System Security Testing

The system security test phase was designed to demonstrate particular security features of the model communications system. It was not intended to prove that the security of the system that was developed could not be violated. One of the major results noted was the fact that no matter how secure a system is, it can still be violated by generating application programs which misroute information obtained through the security kernel. For example, if a corrupt system manager is allowed to modify the encryption key, he could potentially insert a key which had also been passed to someone who is monitoring system external communications. This would allow him to decrypt the outgoing message and compromise any information contained in it. Another example would be if the user was allowed to specify the classification of his outgoing message. This would allow a corrupt user to improperly downgrade information and send it to an unauthorized destination. Tight restrictions imposed by the project manager are required to limit access to the application code segment and prevent these types of problems.

The Gemini system used in this research does not currently support the attachment of classified serial I/O ports. This means that the system does not identify the eight ports in terms of a specific access level. Future versions will be able to identify each port with a specific access class. This will prevent an unclassified user from transmitting data through an unclassified port. User terminals will also be attached at a predefined level to prevent the system manager from creating a classified terminal process at an unclassified port.

Security testing consisted of two major areas. First, communications were established between users having the same access level. Messages were passed between the two terminals via the multilevel communication process. Initially a multilevel (min-unclass, max-confidential) system manager was created to coordinate communications between two unclassified users. As discussed above, the Gemini system does not currently support secure serial I/O which necessitated manually entering the access level of each of the terminal processes. Once the secure serial I/O capability is available, the system manager would not have to specify the classification of data going to and from the remote user terminals because it is already specified by the system classification of the port to which the terminal is attached. Following unclassified testing, confidential messages were exchanged between the simulated confidential user terminals.

The second task was to test the system's ability to detect and respond to a security violation. To accomplish this, the user terminals were assigned different access levels. When messages were sent between the terminals, the system recognized the security violation and issued the appropriate error message back to the originator. In this case the security check consisted of a comparison of the incoming message header, with the system manager defined destination access level. The error message interrupts the normal sequential passing of messages, to inform the originator that the destination of his message did not have the proper access level to receive it.

Although only one communication process was used to create both terminal processes, the terminals operate independently to simulate being located at two different activities. They send and receive messages from different physical ports, and communicate to each other using different external communication ports. Inter-process synchronization was accomplished by allowing only one terminal to send a message at a time. Once a terminal's message transmission was complete, control was passed to the other terminal to allow it to display the incoming message, and send its outgoing message. This technique was chosen to facilitate testing, and is not the only method which could have been used. Depending on the particular application, a timed polling scheme with all terminals operating simultaneously may be appropriate.

3. <u>Encryption Testing</u>

All data passed between the external communications ports was encrypted using the Data Encryption Standard (DES) algorithm operating in the cipher block chaining (CBC) mode. Data encryption was enhanced by using the techniques discussed in Chapter II. The objective was to create a unique ciphertext for each transmitted message, regardless of whether the actual text of the message was the same. This was accomplished by providing the data encryption device with a unique initialization vector for each message. This system uses the transmission time of the message as the initialization vector. In an actual system, this would need to be modified by a random offset to prevent someone monitoring the outgoing traffic from gaining access to the initialization vector. Another way to do this would be to use a sequential message number which had been randomly modified as the initialization vector. As long as the initialization vector is unique, no two messages will have the same ciphertext.

To test the data encryption device using the data encryption techniques discussed above, a series of test messages were generated. These messages were used to test specific features of the data encryption process. The system manager application program was modified to display the ciphertext of each encrypted block. Identical messages were transmitted to compare the resulting ciphertexts. As expected, the resulting ciphertexts were not the same. Error propagation was also tested by inserting errors in the received ciphertext prior to decryption. The errors appeared in the decrypted text however were confined to the block of data in which the error was introduced.

Another area of concern was that encrypting the outgoing message adversely effect system operation. As discussed in Chapter II, there are two basic encryption methods. They are the methods which utilize feedback to provide added security such as the cipher block chaining (CBC) method, and those which do not, such as the electronic code book (ECB) method. In the Gemini system, use of a feedback mode requires that the encryption and decryption devices be reattached with the new feedback key for each block of data processed. This slow-down could degrade impact system performance where large messages are required to be transmitted at high speeds. A decision would have to be made whether to sacrifice some security by using the ECB mode in order to gain speed. This potential problem is largely overcome in the Gemini system by the speed of the Intel APX-286 microprocessor. When test strings were used to provide continuous output on the external communications ports, no noticeable slow-down was observed when using the CBC (feedback) mode.

V. CONCLUSIONS

In this thesis a model secure communications system was developed to demonstrate the feasibility of using a multilevel secure computer system as a secure front end for data communications in an office to office communication environment. The Gemini Trusted Multiple Microcomputer Base used in this research proved to be an extremely flexible system, easily capable of providing a high speed data communication interface. The following observations concern the use of a multilevel secure computer system in this capacity:

1) The major advantages of a multilevel secure front end are the reduction in the message transmission delay due to internal and external processing requirements, and the additional flexibility it provides in developing a discretionary security policy. Each security classification can be broken into several 'need to know' classes which further restrict access to information, and provide additional security.

2) By developing secure application software which automates internal message routing, and security record keeping requirements which are currently done manually, a significant reduction in the manhours required to process and store sensitive information can be realized.

3) A major problem in developing application software is the difficulty encountered in generating programs to run in the secure environment. There is currently no way of taking existing software for a particular system, and directly adapting it to run in a secure environment.

4) Electronic transmission of sensitive information in an encrypted format, reduces the delay associated with traditional transmission techniques. Information which can not be readily converted to a form which could be transmitted electronically, would still be transmitted via conventional routes.

5) Data encryption can be used to greatly increase the protection of transmitted data, without adversely effecting system performance. Although not currently approved for transmission of Department of Defense (DOD) classified data, the Data Encryption Standard (DES) algorithm when used in cipher block chaining (CBC) mode as discussed in Chapter II provides the maximum protection. By multiprocessing the DES encryption process with a DOD approved method the system can be used for transmission of classified data.

6) By directly controlling the access of remote users to external communication devices, the security manager

can have positive control over all incoming and outgoing messages. The security manager defines the access level of each device, preventing unauthorized transmission of classified data, and ensuring that incoming traffic is routed in a secure manner.

7) The multilevel secure computer system can interface directly with a wide variety of communication equipment, however, incompatible devices would still have to be monitored separately. It is important to note that, the use of a multilevel secure computer system does not necessarily reduce the physical security requirements. Physical devices must still be provided protection consistent with the classification level of the information they process.

As the number of computerized processing systems with external communications capabilities grows, the need to have a trusted secure interface between system users and external communications devices becomes increasingly important. Use of a multilevel secure computer system as a secure front end interface can greatly enhance overall system security. Functioning as both an external communications interface and internal traffic manager, the trusted computer system provides the project manager with centralized control over access and distribution of sensitive information.

APPENDIX A
TERMINAL UTILITY PROGRAM LISTING

The terminal utility program is compiled and prepared for execution in almost the same manner as the system manager application program discussed in Appendix A. By modifying certain parameters which are identified in the program listing, the system manager can specify the physical port and terminal number of the remote terminal process. Once copied to the bootable disk which includes the operating system generation (SYSGEN) program, the tl-util.cmd and t2-util.cmd files are automatically entered in the secure volume created using the sysmgr.ssb file. To enter additional terminals, the sysmgr.ssb file would have to be modified to specify the entry number of the new terminal utility program. A listing of the tl-util.kmd file which is used to create the tl-util.cmd file is included following the terminal utility source code listing.

```
{****************************************************************

        program name: t1_util.txt

        date: 18 feb 86

        author: P. J. Corbett Lt./USN

        for: AEGIS Modelling Group

        advisor: Prof. Kodres

        purpose: This program is initiated when a terminal
process is created by the system manager process
(sysmgr.txt).  It allows the terminal operator to enter
and send messages via the sysmgr process, as well as dis-
play incoming messages.  Message specifications and
terminal access level are determined by the sysmgr process
and passed to the terminal process in its r1_process_def
record.  Other system constants are provided in the
mgr-typ.zli and mgr-con.zli files.  Incoming and outgoing
buffers are used to store messages.  Eventcounts for these
segments are used to synchronize system communications.

****************************************************************}

module t1_util;

const


        { system constant include files }

        {$i gate-con.zli}

        {$i r1-con.zli}
        {$i mgr-con.zli}


        t_phys_dev = 5;           { physical port to which  }
        term_num = '1';           { terminal is attached    }

type


        { common type include files }
        {$i gate-typ.zli}

        {$i lit-typ.zli}
        {$i rlp-typ.zli}
        {$i mgr-typ.zli}

        { library procedure include files }

         {$i lib.zli}

         {$i io.zli}

         {$i gate.zli}

        {$i seg-mgr.zli}

{--------------------------------------------------


        proc_name: input_mess

        purpose:  This procedure allows the operator
to input a message into the outgoing message buffer.
Characters are input into 8 byte blocks so that they
will be compatible with the data encryption device.
The character '$' is used to indicate the end of the
message.  The intial block is reserved for the
address header.  Format of the header is as follows:

        blk[1][1]: source
        blk[1][2]: destination
        blk[1][3-5]: message number
        blk[1][6]: classification
        blk[1][7-8]: number of blocks in msg

Source,destination, and number of blocks are entered
in this procedure.  Remaining entries are filled in
by the sysmgr process prior to transmission.

--------------------------------------------------}

procedure input_mess(comn_buf:integer;
                        var buf_stat:boolean);
```

```
var
        mess_rec: buf_rec;

        blk_cnt: string;
        temp_ptr,input_ptr: pointer;
        charin:char;
        i,j,k,success:integer;
        count: integer;
        proc_suc: boolean;



begin {input_mess}



{ create pointer to start of input buffer }


input_ptr:=lib_mk_pntr(ldt_table,comn_buf,1);


temp_ptr:=input_ptr;

clr_screen(proc_suc);

putln(w_dev,'enter message to be transmitted');
putln(w_dev,'enter a $ to indicate end of msg');
putln(w_dev,' ');


        { initialize msg block counter }
         i:=1:

{ begin character entry loop }
while(charin <> '$') and
                (i <> mess_buf_size+1) do begin

        { block 1 is addr, block 2 is strt of msg }
        i:=i+1;

        { begin loop to read 8 char for each block }
        for j:= 1 to 8 do begin

                if charin <> '$' then begin

                getchar(r_dev,charin);
                mess_rec.block[i][j]:= charin;
```

```
                    { echo character input }
                    putchar(w_dev,charin);

                    end else

                    { if charin='$' then pad the remain-
                    ing entries with '$' to avoid sending
                    an incomplete block                }
                    mess_rec.block[i][j]:= '$';


          end; {for}


end; {while}

{ insert sentinel at end of buffer in case input
buffer size was exceeded }
mess_rec.block[mess_buf_size][8]:='$';


{ count keeps track of number of blocks input }
count:=i;

{ fillin address block source,dest,and num_blk }
mess_rec.num_blk:=count;
mess_rec.block[1][1]:= term_num;

putln(w_dev,' ');
putln(w_dev,'enter destination terminal number');
getchar(r_dev,mess_rec.block[1][2]);
putchar(w_dev,mess_rec.block[1][2]);

putln(w_dev,' ');

binascii(count,3,blk_cnt,'0');

for i:= 1 to 2 do
        mess_rec.block[1][i+6]:= blk_cnt[i];



{ place mess_rec in outgoing message buffer to
await transmission }
move(mess_rec,temp_ptr^,sizeof(mess_rec));

buf_stat:=true;

putln(w_dev,'message input complete');
```

```
end; {input_mess}


{-------------------------------------------------------------

        proc name: xmit_mess

        purpose:  This procedure alerts the sysmgr
process that the operator desires to transmit the
message stored in the outgoing message buffer. This
is done by advancing the outbuf eventcount.  The
sysmgr process notifies the terminal process that
the message has been sent by advancing the inbuf
eventcount.

-------------------------------------------------------------}

procedure xmit_mess(inbuf_slot:integer;outbuf_slot:integer;
                var inbuf_evc:integer;
                var xmit_buf_stat:boolean);

var

        success:integer;

begin {xmit_mess}



        { notify sysmgr, msg ready to xmit }
        advance(outbuf_slot,success);
        show_err('outbuf advance error',success);

        { await sysmgr xmit complete notification }
        await(inbuf_slot,inbuf_evc+1,success);
        show_err('await inbuf error',success);

        inbuf_evc:=inbuf_evc+1;

        putln(w_dev,'message transmission complete');

        xmit_buf_stat:=false;

end; {xmit_mess}

{-------------------------------------------------------------
```

```
        proc name: disp_mess

        purpose:  This procedure displays the
message stored in the incoming buffer segment.  It
is similar in structure to input_mess.

------------------------------------------------------------}

procedure disp_mess(comn_buf:integer;
                        var rec_buf_stat:boolean);

var

        disp_rec:buf_rec;
        disp_ptr:pointer;
        d_char:char;
        i,j:integer;
        proc_suc: boolean;

begin {disp_mess}


{ create pointer to incoming message buffer segment}
disp_ptr:= lib_mk_pntr(ldt_table,comn_buf,1);

clr_screen(proc_suc);

putln(w_dev,'begin display of received message');


{ place contents of incoming message buffer into
disp_rec }
move(disp_ptr^,disp_rec,sizeof(disp_rec));

{ check incoming message for error message which
is indicated by a source of 'Ø' }

{ if no error message then begin display }
if disp_rec.block[1][1] <> 'Ø' then begin

        putstr(w_dev,'message from terminal');
        putchar(w_dev,disp_rec.block[1][2]);
        putln(w_dev,' ');
        putln(w_dev,'message follows --');

        putln(w_dev,' ');

       i:=1;
```

```
        { output inbuf contents to terminal }
        while (d_char <> '$') and
                (i <> mess_buf_size+1) do begin

        for j:= 1 to 8 do begin

                if d_char <> '$' then begin

                putchar(w_dev,disp_rec.block[i][j]);

                d_char:=disp_rec.block[i][j];

                end; {if}


        end; {for}

        i:=i+1;

       end; {while}

end else begin

        putln(w_dev,'message from system manager');
        putln(w_dev,'security violation');
        putln(w_dev,'improper dest access');
        putln(w_dev,'message not delivered');

end; {if}

putln(w_dev,'end of message');

rec_buf_stat:=false;

end; {disp_mess}
```

```
{--------------------------------------------------------------

        proc name: logoff

        purpose:  This procedure disatles the term
and makes the resources assigned tc the terminal
process availatle.  No new terminal process is
created to replace it.


--------------------------------------------------------------}


procedure logoff(init:r1_process_def);


var
        suc,success:integer;

tegin {logoff}

 putln(w_dev,'terminating child segments');
{ to reinitialize a terminal process at this term,
process segments would have to te terminated prior
to the self_delete call }

putln(w_dev,'self deleting child process now');
putln(w_dev,'terminal off-line');

detach(w_dev);
detach(r_dev);

self_delete(init.initial_seg[stack_offset],success);

        if (success <> no_error) then tegin
                attach(t_phys_dev,w_dev,false,suc);
                attach(t_phys_dev,r_dev,true,suc);
                show_err('child self delete error',success);
        end; {if}

end; {logoff}
```

```
{-----------------------------------------------------------

        proc name: show_err

        purpose:  This procedure is called to
display the success code of the resource mngmnt
call if it is other than zero.  If the success code
indicates no_error then no message is output.

-----------------------------------------------------------}

procedure show_err(str:string; code:integer);

begin {show_err}

        if code <> no_error then begin
                putstr(w_dev,str);
                putstr(w_dev,' ');
                putdec(w_dev,code);
                putln(w_dev,'  ');
        end;
end; {show_err}

{---------------------------------------------------

        proc name: clr_screen

        purpose: Clears display screen.

---------------------------------------------------}

procedure clr_screen(proc_suc:boolean);

var

        i:integer;

begin {clr_screen}

for i:= 1 to 25 do
        putln(w_dev,' ');

end; {clr_screen}
```

```
{*****************************************************
*****************************************************


        proc name: main

        purpose:  This procedure provides a mode
selection menu for the terminal operator.  It moni-
tors buffer status and calls the appropriate proc
dependent on the mode selection entry.


**************************************************
**************************************************}
procedure main( var init : r1_process_def );

var
    success : integer;
        seg_num:integer;
        mode:char;
        xmit_buf_stat,rec_buf_stat:boolean;
        temp_str:string[1];
        i,level:integer;
        inbuf_evc: integer;
        stk_evc:integer;
        sys_start:boolean;


begin {main}

        { initialize terminal process parameters }
        xmit_buf_stat:=false;
        mode:='0';

        { sys_start=false for twerminal 1 only all
        other terminals should have sys_start=true}
        sys_start:=false;
        inbuf_evc:=0;

        { attach terminal as read/write device }
        attach(t_phys_dev,w_dev,false,success);
        attach(t_phys_dev,r_dev,true,success);
        show_err('attach term read device error',success);

        putln(w_dev,' ');
        putln(w_dev,'terminal active term number');
        putchar(w_dev,term_num);
        putln(w_dev,' ');
        { stack eventcount is advanced to notify
        sysmgr that terminal is activated }
        advance(init.initial_seg[stack_offset],success);
        show_err('stack advance error',success);


{ loop until operator enters 'e' to indicate logoff }
while mode <> 'e' do begin

{ inbuf_evc is used to have the terminal wait after
transmitting a message until a reply is received
from the dest term.  It is initially advanced for
terminal 1 to start the system and then is advanced
upon receipt of an incoming message }
await(init.initial_seg[inbuf_offset],inbuf_evc+1,success);
show_err('await incoming message',success);

inbuf_evc:=inbuf_evc+1;

{ sys_start is used to avoid the 'display incoming
message' prompt at terminal 1 when the system is
started.  Once the system is operating it will
always be true }
if sys_start= true then begin

rec_buf_stat:=true;
putln(w_dev,'display incoming message');

end; {if}


sys_start:=true;


        { inner loop is used to indicate that a
        message has been sent and alert the operator
        that the terminal is waiting for a reply}
        while mode <> 'x' do begin

        { help menu consists of a display of term
        access level, and a display of possible
        modes }
                putstr(w_dev,'terminal compromise level');
                level:= init.root_access.compromise[1];

                case level of

                        0: putln(w_dev,'unclassified');
                        2: putln(w_dev,'confidential');
                4: putln(w_dev,'secret');
                6: putln(w_dev,'top secret');

        end; {case}



putln(w_dev,'enter mode desired');
putln(w_dev,'i= input message');
putln(w_dev,'d= display received message');
putln(w_dev,'x= transmit message');
putln(w_dev,'e= logoff');
putln(w_dev,' ');
putstr(w_dev,'enter mode here');

getchar(r_dev,mode);

if mode= 'i' then begin

        if xmit_buf_stat= false then begin

        { enter message to be stored in
        outgoing message buffer }
        input_mess(init.initial_seg[outbuf_offset],
        xmit_buf_stat);

        end else begin

        putln(w_dev,'message waiting to be xmit');
        end;


end else if mode= 'd' then begin


        if rec_buf_stat= true then begin

        putln(w_dev,'entering display module');
        { display contents of incoming
        message buffer }
        disp_mess(init.initial_seg[inbuf_offset],
                        rec_buf_stat);


        end else begin

        putln(w_dev,'incoming buffer empty');
        end;
        end else if mode= 'x' then begin

                if xmit_buf_stat= true then begin

                putln(w_dev,'sending message to be xmit');
                xmit_mess(init.initial_seg[inbuf_offset],
                        init.initial_seg[outbuf_offset],
                        inbuf_evc,xmit_buf_stat);

                end else begin

                putln(w_dev,'outgoing buffer empty');

                end;

        end else if mode='e' then begin

                putln(w_dev,'logoff process initiated');
                logoff(init);

        end else

                putln(w_dev,'mode entry error try again');

        { end of inner loop-exit after msg xmit }
        end; {while}

putln(w_dev,'waiting for incoming traffic');
{ reset mode selection value }
mode:='0';


end; {while}
```

```
    putln(w_dev,'end of terminal 1 process');
    detach(w_dev);
    detach(r_dev);

{infinite loop to avoid crash}

        while true do;

end; {main}


modend.
```

```
{*************************************************************

        program name: t#-util.kmd

        author: P.J. Corbett, Lt., USN
        date: 28 Feb 86
        purpose:  This program is used when linking the
terminal utility program after it has been successfully
compiled.  It eliminates the need to enter the names of
the modules the program is to be linked with each time
a new version of the program is compiled.

        note: the actual t#-util.kmd file  contains only
one line.  Any additional information will cause an
error when the pascal MT+ linker is called.  To adapt this
file for a specific terminal the '#' in the program name
is changed to the terminal number, t1-util, t2-util etc..

*************************************************************}


b:t#-util=b:r1-init,b:t#-util,b:r1lib/s,b:cc/s,paslib/s/p:80
```

APPENDIX B
SYSTEM MANAGER PROGRAM LISTING

The source code for the system manager application program (sysmgr.txt) is written in Pascal MT+. With the exception of the mgr-typ.zli and mgr-con.zli files, all included files are library utility programs which were delivered with version 1.3 of the Gemini operating system. Information concerning how to invoke library functions is contained in [Ref. 14]. Once the text file is compiled, the required modules are linked together by using the sysmgr.kmd submit file with the Pascal MT+ linker. A listing of this file is provided immediately following the source code listing. Upon completion of the linking process, the resulting sysmgr.cmd file must be prepared to run in the Gemini Secure Operating System (GEMSOS) environment. This is accomplished by transferring the sysmgr.cmd file to the bootable disk which contains the operating system generation (SYSGEN) program. Procedures for running the sysgen program are contained in [Ref. 17]. A listing of the submit file sysmgr.ssb which contains the application segment hierarchy used in the system generation process is included at the end of this appendix. Once a secure volume is created on the disk, the Gemini system is reinitialized using the secure volume. This begins execution of the system manager application segment.

```
{--------------------------------------------------------------

        program name- sysmgr.txt

        date: 18 feb 86

        author: P. J. Corbett Lt./USN

        for: AEGIS Modeling Group

        advisor: Prof. Kodres

        purpose:  This program is initialized as a multi-
level process which allows the sysmgr to configure and
operate a multi terminal communication system.  It relies
on information contained in the mgr-con.zli and mgr-typ.zli
files as well as interactive inputs to determine config-
uration parameters.  Once initialization is complete, the
system runs independently allowing remote terminal users
to transmit messages via the multilevel secure front end
process.

--------------------------------------------------------------}

module sysmgr;

{ constant include files }

const

        {$i mgr-con.zli}
        {$i gate-con.zli}
        {$i r1-con.zli}

{ type include files }
type

        {$i gate-typ.zli}
        {$i r1p-typ.zli}
        {$i lib-typ.zli}
        {$i kst-typ.zli}
        {$i mgr-typ.zli}

{ library include }

        {$i io.zli}
        {$i gate.zli}
        {$i seg-mgr.zli}
        {$i cc.zli}
        {$i lib.zli}


{--------------------------------------------------------

        proc name: parm_input

        purpose:  This procedure allows the
sysmgr to input system parameters necessary to test
system operation.


--------------------------------------------------------}

procedure parm_input(var sys_rec:sysmgr_rec;
                         proc_suc:boolean);

var

        temp_str:string;
        temp_char:char;
        temp_int:integer;
        i:integer;

begin {parm_input}


putln(w_dev,'begin entering system parameters');

putln(w_dev,'enter physical port 1 for external comm');
getchar(r_dev,temp_char);
putchar(w_dev,temp_char);

sys_rec.comm_port[1]:=ord(temp_char)-48;
putln(w_dev,' ');

putln(w_dev,'enter physical port 2 for external comm');
getchar(r_dev,temp_char);
putln(w_dev,temp_char);

sys_rec.comm_port[2]:=ord(temp_char)-48;
putln(w_dev,' ');



sys_rec.ch_size:=400;

putln(w_dev,'child size is');
```

```
putdec(w_dev,sys_rec.ch_size);
putln(w_dev,' ');

putln(w_dev,'buffer size is 100 bytes');

sys_rec.b_size:=100;
putln(w_dev,' ');

putln(w_dev,'enter terminal access level');
putln(w_dev,'unclass=0');
putln(w_dev,'conf=2');
putln(w_dev,'secret=4');
putln(w_dev,'ts=6');

putln(w_dev,'entry must be within sysmgr access range');

for i:= 1 to num_term do begin

        putstr(w_dev,'terminal');
        putdec(w_dev,i);
        putstr(w_dev,'access level is');

        getchar(r_dev,temp_char);
        putchar(w_dev,temp_char);
        temp_int:=ord(temp_char)-48;

        { fill access_class record with entered class }
        fillchar(sys_rec.ch_access[i],sizeof(access_class),
                         chr(temp_int));

        putln(w_dev,' ');

end; {for}

putln(w_dev,'enter 8 character crypto key (no echo)');

for i:= 1 to 8 do begin

        getchar(r_dev,temp_char);
        sys_rec.key[i]:=ord(temp_char)-48;

end; {for}

putln(w_dev,'crypto key inserted');

{ fixed parameters }

{ code segment entry numbers }
```

```
sys_rec.chld_ent[1]:=6;
sys_rec.chld_ent[2]:=7;

putln(w_dev,'parameter entry complete');
end; {parm_input}

{-----------------------------------------------------

        proc name: sys_config

        purpose:  This procedure configures the external
communication ports identified in parm_input for port to
port communications with flow control.  They are attached
to read and write sequentially 8 bytes at a time to be
compatible with the data encryption device.


-----------------------------------------------------}

procedure sys_config( send_port: integer;
                      recv_port: integer;
                      var config_suc:boolean);

var

        rd_dev,wr_dev: dev_name;
        rd_parm,wr_parm: dev_parm_rec;
        success: integer;

begin { sys_config }

config_suc:= false;

putln(w_dev,'configure transmit and receive ports');

{ attach xmit and recv ports for computer to computer
communications }

{ fill-in attach_device calling arguments }

{ receiver should be attached first }
rd_dev.name:= sior;
rd_dev.num:= recv_port;
rd_dev.d_type:= io;

rd_parm.sior.mr1:= $04d;        { device mode entries }
rd_parm.sior.mr2:= $03e;
rd_parm.sior.io_mode:= rts_oflow;
rd_parm.sior.max:= 8;
rd_parm.sior.delim_active:= false;

attach_device(rd_dev,recv_slt,rd_parm,success);
show_err(' receiver attach error',success);

{ attach transmitter }
wr_dev.name:= siow;
wr_dev.num:= send_port;
wr_dev.d_type:= io;

wr_parm.siow.mr1:= $Ø4d;          { device mode entries }
wr_parm.siow.mr2:= $Ø3e;
wr_parm.siow.io_mode:= asrt_none;

attach_device(wr_dev,xmit_slt,wr_parm,success);
show_err('transmitter attach error',success);

putln(w_dev,'comm devices attached');

{ xmit and recv attached computer to computer no flow
control }

config_suc:= true;

end; {sys_config}


{-----------------------------------------------

        proc name: comm_tst

        purpose:  This procedure checks communications in
both directions by transmitting a test string of data. Once
communications have been checked the comm devices are
detached.

-----------------------------------------------}

procedure comm_tst(init: r1_process_def;
                   send_port: integer;
                   recv_port: integer;
                   var comm_tst_suc: boolean);

var

charin,charout: array [1..8] of char;
wr_class,rd_class: access_class;
i,success: integer;
size: integer;
```

```
begin { comm_tst }

comm_tst_suc:= false;


putln(w_dev,'begin comm test');


{ transmitter access_class for comm test }
wr_class.compromise:= init.resources.max_class.compromise;
wr_class.integrity:= init.resources.min_class.integrity;

putstr(w_dev,'outgoing string is ');

for i:= 1 to 8 do begin

        charout[i]:= 't';
        putchar(w_dev,charout[i]);

end; {for}

putln(w_dev,' ');

write_sequential(xmit_slt,addr(charout),8,wr_class,success);
show_err('write sequential error',success);

read_sequential(recv_slt,addr(charin),size,wr_class,success);
show_err('read sequential error',success);

for i:= 1 to 8 do

        putchar(w_dev,charin[i]);

putln(w_dev,' ');

detach_device(xmit_slt,success);
show_err('transmitter detach error',success);

detach_device(recv_slt,success);
show_err('receiver detach',success);

putln(w_dev,'comm test complete');

comm_tst_suc:= true;

end; { comm_tst }
```

```
{-------------------------------------------------------

        proc name: att_crypto

        purpose:  This procedure uses four process local
device slots to attach the required encryption and
decryption devices.  Crypto key and feedback key are pro-
vided in the procedure call.  Devices are attached using
the cipher block chaining (CBC) mode.


--------------------------------------------------------}

procedure att_crypto(cry_key: buf8;
                cry_fbkey: buf8;
                var att_crypto_suc: boolean);

var

rendev,wendev,rdedev,wdedev: dev_name;
ren_parm,wen_parm,wde_parm,rde_parm: dev_parm_rec;
success: integer;

begin { att_crypto }

att_crypto_suc:= false;


{ attach read encryption device }

        rendev.name:= dcp_ren;
        rendev.num:= 0;
        rendev.d_type:= io;

        ren_parm.ren.blk_size:= 8;      { 8 bytes per blk }

attach_device(rendev,ren_slt,ren_parm,success);
show_err('attach ren device error',success);

{ attach write encryption device }

        wendev.name:= dcp_wen;
        wendev.num:= 0;
        wendev.d_type:= io;

        wen_parm.wen.mode:= 1;          { 1 for CBC mode }
        wen_parm.wen.key:= cry_key;
        wen_parm.wen.fb_key:= cry_fbkey;
```

```
attach_device(wendev,wen_slt,wen_parm,success);
show_err('attach wen device error',success);


{ attach read decryption device }

        rdedev.name:= dcp_rde;
        rdedev.num:= 1;
        rdedev.d_type:= io;

        rde_parm.rde.mode:= 1;          { 1 for CBC mode }
        rde_parm.rde.key:= cry_key;
        rde_parm.rde.fbkey:= cry_fbkey;
        rde_parm.rde.blk_size:= 8;

attach_device(rdedev,rde_slt,rde_parm,success);
show_err('attach rde device error',success);

{ attach write decryption device }

        wdedev.name:= dcp_wde;
        wdedev.num:= 1;
        wdedev.d_type:= io;

        { wde_parm is blank record }

attach_device(wdedev,wde_slt,wde_parm,success);
show_err('attach wde device error',success);


att_crypto_suc:= true;

end; { att_crypto }


{----------------------------------------------------

        proc name: crypto_tst

        purpose:  This procedure verifies that the
encryption and decryption devices are working properly.
A test string is encrypted then decrypted using test
keys.  Results are output to the sysmgr terminal.  When
complete all data ciphering devices are detached.


----------------------------------------------------}

procedure crypto_tst(init: r1_process_def;
                        crypto_tst_suc: boolean);


var

encryptin,encryptout,decryptout: array [1..8] of char;
wr_class,rd_class: access_class;
size: integer;
i: integer;
success: integer;
proc_suc: boolean;

begin { crypto_tst }

crypto_tst_suc:= false;

putln(w_dev,'begin crypto device test');

wr_class.compromise:=init.resources.max_class.compromise;
wr_class.integrity:=init.resources.min_class.integrity;

putstr(w_dev,'crypto test string is ');

for i:= 1 to 8 do begin

        encryptin[i]:= 't';
        putchar(w_dev,encryptin[i]);

end; {for}
putln(w_dev,' ');

{ write test string to encryption device }
write_sequential(wen_slt,addr(encryptin),8,wr_class,
                    success);
show_err('wen siow error',success);

{ read encrypted string }
read_sequential(ren_slt,addr(encryptout),size,rd_class,
                    success);
show_err('ren sicr error',success);

putstr(w_dev,'encrypted string is ');

for i:= 1 to 8 do
        putchar(w_dev,encryptout[i]);

putln(w_dev,' ');

{ write encrypted string to decryption device }
```

```
write_sequential(wde_slt,addr(encryptout),8,wr_class,
                 success);
show_err('wde siow error',success);


{ read decrypted string }
read_sequential(rde_slt,addr(decryptout),size,rd_class,
                success);
show_err('rde sicr error',success);

putstr(w_dev,'decrypted string is ');
for i:= 1 tc 8 do
        putchar(w_dev,decryptout[i]);

putln(w_dev,' '):
putln(w_dev,'crypto test complete');

{ detach encryption/decryption devices }
det_crypto(proc_suc);


crypto_tst_suc:= true;

end; { crypto_tst }


{---------------------------------------------------

        proc name: term_proc_create

        purpose:  This procedure creates a single level
child process for a user terminal using the parameters
specified by the sysmgr in parm_input.  The child process
code segment is a terminal utility program which attaches
the child process at the desired physical port.  Four seg-
ments are passed to the child process.  The stack segment
contains the ch_init:r1_process_def record.  The two
common message buffer segments (inbuf and outbuf) are used
to pass messages between the parent and child processes.
Finally a code segment is required for all child processes.
Process local segment numbers as well as pointers to the
message buffer segments are passed back to the main pro-
cedure when the child process has been created.

---------------------------------------------------}
```

```
procedure term_proc_create(init: r1_process_def;
                ch_parm: sysmgr_rec;
                chld_num: integer;
                var stk_slot:ch_array;
                var outbuf_slot: ch_array;
                var inbuf_slot: ch_array;
                var out_ptr: pointer;
                var in_ptr: pointer;
                term_create_suc: boolean);


var

ch_cde_seg_num: ch_array;
pt_seg_num: ch_array;

chld_entry: integer;

ch_init: r1_process_def;
ch_addr_rec: r1_addr_array;
ch_reg_rec: r1_reg_record;
ch_res_rec: r1_res_record;
ch_seg_list: seg_array;

stk_init_ptr: ^r1_process_def;
stk_ptr: var_pointer;
inbuf_ptr,outbuf_ptr: array [1..num_term] of pointer;

seg_mgr_bytes: integer;

stack_size,chld_size,buf_size: integer;
stk_evc_val: ch_array;
size,success: integer;
i,j: integer;
class:access_class;


begin { term_proc_create }

term_create_suc:=false;

{ initialize child parameters }
chld_size:= ch_parm.ch_size;
buf_size:= ch_parm.b_size;
j:= chld_num;
chld_entry:= ch_parm.chld_ent[j];
```

```
{=============================================================
create, makeknown, and swapin as appropriate, child
segments

=============================================================}

{ makeknown terminal utility code segment located at child
entry number specified in sysmgr.ssb file }
seg_makeknown(init.initial_seg[root_offset],chld_entry,
        ch_cde_seg_num[j],r_e,size,class,success);
show_err('makeknown child entry off root error',success);


{ create and makeknown child process base }
seg_create(init.initial_seg[code_offset],chld_num,Ø,success);
show_err('create process base for child error',success);

seg_makeknown(init.initial_seg[code_offset],chld_num,
        pt_seg_num[j],r_w,size,class,success);
show_err('makeknown child process base error',success);


{ determine required size for stack. it must be large
enough required information for child initialization.  This
expression was adapted from the pro-tst.zpa process
creation demonstration program }

seg_mgr_bytes:= sizeof(stack_header)+sizeof(kst_header)+
                (sizeof(kst_entry)*init.num_kst);

stack_size:= r1_stack_size+vect_size+seg_mgr_bytes;

{ create, makeknown, and swapin child stack segment }
seg_create(pt_seg_num[j],Ø,stack_size-1,success);
show_err('create child stack error',success);

seg_makeknown(pt_seg_num[j],Ø,stk_slot[j],r_w,size,
                class,success);
show_err('makeknown child stack error',success);

swapin_segment(stk_slot[j],success);
show_err('swapin child stack error',success);

{ stack eventcount is used to notify sysmgr that the
terminal process is activated.  It is also used as an
entry in the ch_init record }
read_evc(stk_slot[j],stk_evc_val[j],success);
show_err('read stack evc error',success);
```

```
{ create message buffers }

{ outgoing message buffer }
seg_create(pb_seg_num[j],1,buf_size,success);
show_err('create outbuf error',success);

seg_makeknown(pb_seg_num[j],1,outbuf_slot[j],
                r_w,size,class,success);
show_err('outbuf makeknown error',success);

swapin_segment(outbuf_slot[j],success);
show_err('outbuf swapin error',success);

{ incoming message buffer }
seg_create(pb_seg_num[j],2,buf_size,success);
show_err('creat inbuf error',success);

seg_makeknown(pb_seg_num[j],2,inbuf_slot[j],
                r_w,size,class,success);
show_err('inbuf makeknown error',success);

swapin_segment(inbuf_slot[j],success);
show_err('inbuf swapin error',success);

{ fillin ch_seg_list }

{ ch_seg_list determines order in which segments are passed
to the child process }
ch_seg_list[stack_offset]:= stk_slot[j];
ch_seg_list[code_offset]:=ch_cde_seg_num[j];
ch_seg_list[root_offset]:=init.initial_seg[root_offset];
ch_seg_list[outbuf_offset]:= outbuf_slot[j];
ch_seg_list[inbuf_offset]:= inbuf_slot[j];

{fillin child init record }

{ ch_init record is placed on stack for use by child
process when created }
ch_init.cpu:= init.cpu;
ch_init.num_kst:= init.num_kst;
ch_init.root_access:=init.root_access;
ch_init.s_seg:= stack_offset;
ch_init.s_seg_event:= stk_evc_val[j];
{ priority is important in multiprocessing with a single
processor to ensure proper synchronization }
ch_init.resources.priority:= init.resources.priority-10;
lib_integer_to_b24(chld_size,ch_init.resources.memory);
ch_init.resources.processes:= 2;
ch_init.resources.segments:= 90;
{ min_class and max_class determine the access level of
the child process.  Since the terminal process is single
level, they are the same. Levels are specified by the
sysmgr during the parm_input initialization. }
ch_init.resources.min_class:= ch_parm.ch_access[j];
ch_init.resources.max_class:= ch_parm.ch_access[j];
ch_init.sp2:= 0;
ch_init.ring_num:= 1;


{create stack pointer}

{ stack pointer is offset to start of r1_process_def }
stk_ptr.seg:= lib_mk_sel(ldt_table,stk_slot[j],1);
stk_ptr.off:= stack_size-(vect_size+seg_mgr_bytes+
                  sizeof(r1_process_def));
stk_init_ptr:=stk_ptr.p;

{ copy ch_init on to stack }
move(ch_init,stk_init_ptr^,sizeof(r1_process_def));


{ create pointers to message buffers }
{ point to start of message buffer, no offset }

outbuf_ptr[j]:= lib_mk_pntr(ldt_table,outbuf_slot[j],1);
inbuf_ptr[j]:= lib_mk_pntr(ldt_table,inbuf_slot[j],1);


{ fillin remaining records for create_process call }

{ child address record }

{ a maximum of 5 segments may be passed in ch_addr_array }
for i:= 0 to 4 do begin

        ch_addr_rec[i].segment_number:= ch_seg_list[i];


        { code segment must be of type read_execute }
        { others are type read_write                }
        if i = 1 then begin

                ch_addr_rec[i].segment_type:= r_e;

        end else begin

        ch_addr_rec[i].segment_type:= r_w;
        end; {if}

        { swapin allsegments except root_offset }
        if i = 2 then begin

                ch_addr_rec[i].swapin:= false;

        end else begin

                ch_addr_rec[i].swapin:= true;
        end; {if}


        ch_addr_rec[i].protection:= 1;

end; {for}

{ child register record }

ch_reg_rec.ip:= $80;
ch_reg_rec.sp:= stk_ptr.off;
ch_reg_rec.sp1:= stack_size-(vect_size+seg_mgr_bytes);
ch_reg_rec.sp2:= 0;
ch_reg_rec.vec_seg:= 0;
ch_reg_rec.vec_off:=stack_size-vect_size;

{ child resource record }

{ child 1 is located at ch_res_rec.chld_num= 0 }
ch_res_rec.child_num:= chld_num-1;
ch_res_rec.priority:= ch_init.resources.priority;
ch_res_rec.memory:= ch_init.resources.memory;
ch_res_rec.processes:= ch_init.resources.processes;
ch_res_rec.segments:= ch_init.resources.segments;
ch_res_rec.min_class:= ch_init.resources.min_class;
ch_res_rec.max_class:= ch_init.resources.max_class;

{ could not pass array of pointers as calling argument
so had to assign to type pointer variables }
in_ptr:= inbuf_ptr[j];
out_ptr:= outbuf_ptr[j];


putln(w_dev,'creating child process now');

create_process(ch_addr_rec,ch_reg_rec,ch_res_rec,success);
show_err('create child process error',success);
```

```
{ wait for child process to advance stack eventcount
indicating that child process is active }
await(stk_slot[j],stk_evc_val[j]+1,success);
show_err('await stack advance error',success);

putln(wdev,'child process created');

term_create_suc:=true;

end; { term_proc_create }



{-----------------------------------------------------

        proc name: xmit_rec

        purpose:  This procedure takes the message stored
in the outgoing buffer of the source terminal, encrypts
each block, and transmits it sequentially via the
appropriate external communications port.  The crypto-
graphic is provided by the sysmgr_rec.  The fb_key is the
time at which the message is sent. At the receiver the
message is decrypted and stored in the incoming message
buffer of the destination terminal.  Access levels of the
msg and dest are compared.  If they do not match the msg
is not delivered, and an error msg is returned to the
source.

note: fb_key needs to be unique to avoid avoid creating
identical cipher texts when a message is transmitted more
than once.  Time of transmission may not work in applica-
tions where there is a significant time delay in trans-
mission.



-----------------------------------------------------}


procedure xmit_recv( ch_parm: sysmgr_rec;
                orig_term: integer;
                dest_term: integer;
                o_out_ptr: pointer;
                o_in_ptr:pointer;
                d_out_ptr: pointer;
                d_in_ptr: pointer;
                var int_mess_num:integer;
                var recv_suc: boolean);
```

```
var

out_rec,in_rec: buf_rec;
in_ptr,out_ptr: array [1..num_term] of pointer;
time: cc_array;
fb_key: buf8;
srce,dest: char;
int_dest: integer;
str_mess_num:string;
encryptout,decryptin,decryptout: array [1..8] of char;
i,j: integer;
size: integer;
recv_class,xmit_class,class: access_class;
count: integer;
success: integer;
dest_comp,mess_comp: integer;
proc_suc: boolean;

begin { xmit_recv }

recv_suc:= false;

putln(w_dev,'entering transmit/receive module');

sys_config(orig_term,dest_term,proc_suc);

{ retrieve message stored in originator's outgoing msg buf}
move(o_out_ptr^,out_rec,sizeof(out_rec));

{ fill in remaining address block entries }

{ outgoing message number }
out_rec.num:=int_mess_num;

{ message classification }
out_rec.block[1][6]:=chr(ch_parm.ch_access[orig_term].
                compromise[1]+48);

{ insert message number in address block }
binascii(int_mess_num,4,str_mess_num,'0');
for i:= 1 to 3 do
        out_rec.block[1][i+2]:=str_mess_num[i];

{ increment message number counter }
int_mess_num:=int_mess_num+1;
```

```
{ determine fb_key }
{ feedback key is the time of transmission }
{ this provides a unique initialization vector }

{ attach callender clock device }
cc_r_attach(cc_slt,success);
show_err('clockread attach error',success);

{ read calender clock }
cc_r_dev(cc_slt,time,success);
show_err('read time error',success);

{ transmission time = fb_key }
putln(w_dev,'crypto key is');
for i:= 1 to 8 do begin
        fb_key[i]:= ord(time[i+3]);
        putdec(w_dev,fb_key[i]);
end; {for}

detach_device(cc_slt,success);
show_err('clock detach error',success);

{ transmitter access class }
xmit_class:= ch_parm.ch_access[orig_term];


{***********************************
begin transmit/receive loop }

for i:= 1 to cut_rec.num_blk do begin

{ in cbc mode crypto devices must be reattached to transmit
each block.  this is required because the previous encrypt-
ed block is used as the fb_key to encrypt the next block.}

att_crypto(ch_parm.key,fb_key,proc_suc);

{ write to encryption device }
write_sequential(wen_slt,addr(out_rec.block[i]),8,
                 xmit_class,success);
show_err('wen siow error',success);

{ read encrypted text }
read_sequential(ren_slt,addr(encryptout),size,class,success);
show_err('ren sior error',success);
```

```
{ transmit encrypted block }
write_sequential(xmit_slt,addr(encryptout),8,xmit_class,
                    success);
show_err('transmit error',success);

{ determine fb_key for next block }
for j:= 1 to 8 do
         fb_key[j]:=encryptout[j];
putln(w_dev,' ');
{*************************************************************
begin receiving message }

{ receiver access class }
recv_class:= ch_parm.ch_access[dest_term];

{ read encrypted text }
read_sequential(recv_slt,addr(decryptin),size,class,success);
show_err('receive error',success);

putln(w_dev,'received text is');
for j:= 1 to 8 do
         putchar(w_dev,decryptin[j]);
putln(w_dev,' ');

{ write to decryption device }
write_sequential(wde_slt,addr(decryptin),8,recv_class,
                    success);
show_err('wde siow error',success);

{ read decrypted text }
read_sequential(rde_slt,addr(decryptout),size,class,success);
show_err('rde sior error',success);

putln(w_dev,'decrypted text is');
for j:= 1 to 8 do begin
         in_rec.block[i][j]:= decryptout[j];
         putchar(w_dev,decryptout[j]);
end;
putln(w_dev,' ');


{ count is number of blocks in received message }
count:=count+1;

{ detach crypto devices to prepare for next block }
det_crypto(proc_suc);

end; {for}
```

```
detach(xmit_slt);
detach(recv_slt);

{*************************************************************
message transmitted and received }

{ insert number of blocks into incoming record }
in_rec.num_blk:= count;

{ decode address block }
srce:= in_rec.block[1][1];
dest:= in_rec.block[1][2];
putstr(w_dev,'dest is');
putchar(w_dev,dest);
putln(w_dev,' ');

int_dest:= ord(dest)-48;
putstr(w_dev,'int_dest is');
putdec(w_dev,int_dest);
putln(w_dev,' ');

dest_comp:= ch_parm.ch_access[int_dest].compromise[1];

mess_comp:= ord(in_rec.block[1][6])-48;
putln(w_dev,' ');

putstr(w_dev,'dest_comp-mess_comp');
putdec(w_dev,dest_comp);
putdec(w_dev,mess_comp);
putln(w_dev,' ');

{ compare message and destination access levels for
possible security violation }
if mess_comp <> dest_comp then begin

        { if srce= 'Ø' then incoming message is an error
        message concerning a security violation }
        if srce <>'Ø' then begin

        putln(w_dev,'security violation message number');

        for i:= 3 to 5 do
                putchar(w_dev,in_rec.block[1][i]);
                recv_suc:=false;

                { prepare error msg for transmission }
                err_msg(srce,d_out_ptr,proc_suc);
        end else begin

                { if incoming traffic is an error msg then
                move it to the incomming message buffer of
                the destination terminal }
                move(in_rec,d_in_ptr^,sizeof(in_rec));

                { reset recv_suc }
                recv_suc:=true;

        end;

end else begin

        { if no violation, move msg into incoming msg
        buffer of destination terminal }
        move(in_rec,d_in_ptr^,sizeof(in_rec));
        recv_suc:=true;

end; {if}


end; {xmit_rec}


{----------------------------------------------------

        proc name: err_msg

        purpose:  In the event of a security violation,
this procedure fills destination outgoing buffer with
an error message.  This error message is then transmitted
to the source for display at the originator's terminal.


----------------------------------------------------}

procedure err_msg(dest:char;
                xmit_ptr:pointer;
                var err_msg_suc:boolean);
var

        err_rec: buf_rec;
        i:integer;

begin {err_msg}
```

```
err_rec.num:= Ø;
{ error msg has only an address block }
err_rec.num_blk:=1;

{ source of 'Ø' indicates an error message }
err_rec.block[1][1]:='Ø';

err_rec.block[1][2]:=dest;

{ remainder of address block is empty }
for i:= 3 to 8 do
        err_rec.block[1][i]:='Ø';

{ move error message to outgoing buffer of dest term
for transmission back to source }
move(err_rec,xmit_ptr^,sizeof(err_rec));

end; {err_msg}

{--------------------------------------------------

        proc name: det_crypto

        purpose:  This procedure detachs all data
encryption/decryption devices.

--------------------------------------------------}

procedure det_crypto(var proc_suc:boolean);

begin {det_crypto}

        detach(wen_slt);
        detach(ren_slt);
        detach(wde_slt);
        detach(rde_slt);

end; {det_crypto}

{--------------------------------------------------

        proc name: show_err

        purpose:  This procedure is called to display
the success code of the resource management call if it is
other than zero.  If the success code indicates no_error
then no message is output.

--------------------------------------------------}
```

```
procedure show_err(str: string;
                   code: integer);

begin {show_err}

if code <> no_error then begin

         putstr(w_dev,str);
         putstr(w_dev,' ');
         putdec(w_dev,code);
         putln(w_dev,' ');
end;

end; { show_err }

{--------------------------------------------------------------

         proc name: main

         purpose:  This procedure initializes system
operation.  It performs comm and crypto checks and then
creates a single level process for each remote terminal.
Once the system is on-line, it controls access to the
external communications ports.  Messages are transmitted
and received, and security checks are performed on all
incoming traffic.

--------------------------------------------------------------}

procedure main( var init : r1_process_def);

var

         i: integer;
         stk_slt,bufout_slt,bufin_slt: ch_array;
         bufout_ptr,bufin_ptr: array [1..num_term]
                                    of pointer;
         bufout_evc,bufin_evc: ch_array;
         mgr_rec: sysmgr_rec;
         test_key,test_fbkey: buf8;
         mess_dest,mess_srce: integer;
         temp1_port,temp2_port: integer;
         success: integer;
         ch_num: integer;
         proc_suc: boolean;
         recv_suc: boolean;
         mess_num:integer;
```

```
begin {main}

attach(init.cpu,w_dev,false,success);
show_err('attach sysmgr siow error',success);

attach(init.cpu,r_dev,true,success);
show_err('attach sysmgr sior error',success);

putln(w_dev,'sysmgr terminal attached');

{ call procedure to enter system parameters }
parm_input( mgr_rec,proc_suc);

{ xmit/recv ports for comm tst }
temp1_port:= mgr_rec.comm_port[1];
temp2_port:= mgr_rec.comm_port[2];

{ configure xmit/recv ports }
sys_config( temp1_port,temp2_port,proc_suc);

{ test comm channel pass 1 }
comm_tst(init,temp1_port,temp2_port,proc_suc);

{ reconfigure xmit/recv ports to transmit in opposite dir }
sys_config(temp2_port,temp1_port,proc_suc);

{ test comm channel pass 2 }
comm_tst(init,temp2_port,temp1_port,proc_suc);

{ keys for crypto test }
for i:= 1 to 8 do begin

        test_key[i]:= i;
        test_fb_key[i]:= i;

end; {for}

{ attach crypto devices in CBC mode }
att_crypto(test_key,test_fbkey,proc_suc);

{ test crypto devices }
crypto_tst(init,proc_suc);

putln(w_dev,'system initialization complete');
```

```
{ loop to create child process for each remote terminal }
for i:= 1 to num_term do begin

        ch_num:= i;

        { create child process }
        term_proc_create(init,
                mgr_rec,
                ch_num,
                stk_slt,
                bufout_slt,
                bufin_slt,
                bufout_ptr[i],
                bufin_ptr[i],
                proc_suc);

putstr(w_dev,'child process created terminal ');
putdec(w_dev,i);
putln(w_dev,' ');

{ initialize buffer event counts }
bufin_evc[i]:= 0;
bufout_evc[i]:= 0;

end; {for}


{ initial mess_dest is terminal 2 }
mess_dest:= 2;
mess_num:=0;

{ to start system advance inbuf_evc for terminal 1 }
advance(bufin_slt[1],success);
show_err('start system inbuf advance error',success);

{ initialize message receipt success value }
recv_suc:=true;

{*******************************************************

  begin independent system operation loop }

while true do begin


{ inner loop synchronizes terminal to terminal
communications }
for i:= 1 to num_term do begin
```

```
   mess_srce:= i;
   mess_num:=mess_num+1;


   { check for received message security violation }
   if recv_suc = true then begin

   { if no error then wait for next outgoing message }
   await(bufout_slt[i],bufout_evc[i]+1,success);
show_err('await message ready for transmit',success);

   bufout_evc[i]:= bufout_evc[i]+1;

   putln(w_dev,' message ready for transmission');

   { transmit and receive outgoing message }
   xmit_recv(mgr_rec,mess_srce,mess_dest,
   bufout_ptr[mess_srce],bufin_ptr[mess_srce],
   bufout_ptr[mess_dest],bufin_ptr[mess_dest],
   mess_num,recv_suc);


   putln(w_dev,'message sent');

   { notify message source that message was xmit }
   advance(bufin_slt[mess_srce],success);
   show_err('advance source inbuf',success);

   { check for received message security violation }
   if recv_suc = true then begin

           { if no error then notify dest terminal
           to display incoming message }
           advance(bufin_slt[mess_dest],success);
       show_err('advance dest inbuf error',success);

           putln(w_dev,'msg recvd and delivered');

           { new dest term is message srce }
           mess_dest:=i;

   end else begin


           { if security violation did occur then
           transmit error msg back to source.  error
           msg has already been placed in outgoing
           buffer by procedure xmit_recv. }
```

```
                xmit_recv(mgr_rec,mess_dest,mess_srce,
                bufout_ptr[mess_dest],bufin_ptr[mess_dest],
                bufout_ptr[mess_srce],bufin_ptr[mess_srce],
                mess_num,proc_suc);

                putln(w_dev,'error msg transmitted');

                { notify srce of incoming error message }

                advance(bufin_slt[mess_srce],success);
        show_err('notify srce of error advance',success);

        end; {if}

        end else begin

        { if received message had a security violation the
        loop will return control to the message source so
        that he can display the error message }

        { recv_suc = true to allow display of error msg }
        recv_suc:= true;

        end; {if}

    end; {for}

    end; {while}


    putln(w_dev,'program complete');
    while true do;


    end; {main}

    modend.
```

```
{*******************************************************

        program name: mgr-typ.zli

        author: P. J. Corbett, Lt., USN
        date: 28 Feb 86
        purpose:  This file contains type declarations
used in both the system manager and remote terminal utility
programs.  It should be included in the type declaration
sections of both programs.


*******************************************************}



sysmgr_rec = record

        comm_port : array [1..2] of integer;
        b_size : integer;
        ch_size : integer;
        ch_access : array [1..num_term] of access_class;
        chld_cnt : array [1..num_term] of integer;
        key: buf8;

end;

buf_rec = record

        num : integer;
        num_blk: integer;
        block : array [1..mess_buf_size] of
                         array [1..8] of char;

end;

ch_array = array [1..num_term] of integer;

{--------------- end mgr-typ.zli----------------------}
```

```
{*************************************************************
.
        program name: mgr-con.zli

        author: P. J. Corbett, Lt. USN
        date: 28 Feb 86
        purpose:  This file contains global constants
used by both the system manager and terminal utility
programs.  It must be included in the constant
declaration section of both programs.


*************************************************************}


        num_term = 2;
        mess_buf_size = 4;

        xmit_slt = 6;
        recv_slt = 7;

        wen_slt = 2;
        ren_slt = 3;
        wde_slt = 4;
        rde_slt = 5;

        cc_slt = 2;

        stack_offset = 0;
        code_offset = 1;
        root_offset = 2;
        pb_offset = 3;
        outbuf_offset = 3;
        inbuf_offset = 4;

        vect_size = 4;

{----------------- end of mgr-con.zli---------------}
```

```
{**********************************************************

        program name: sysmgr.kmd

        author: P.J. Corbett, Lt., USN
        date: 28 Feb 86
        purpose:  This program is used when linking the
sysmgr program after it is compiled.  It eliminates the
need to manually enter each of the file names each time
a new version of the program is compiled.

        note: the actual sysmgr.kmd file  contains only
one line.  Any additional information will cause an
error when the pascal MT+ linker is called.

**********************************************************}


b:sysmgr=b:r1-init,b:sysmgr,b:r1lib/s,b:cc/s,paslib/s/p:80
```

LIST OF REFERENCES

1. Department of Defense Computer Security Center, Ft. Meade, Md., Report CSC-STD-001-83, DOD Trusted Computer System Evaluation Criteria, 15 August 1985.

2. Department of the Navy, OPNAVINST 5239.1A (draft copy), Department of Defense Trusted Network Evaluation Criteria, 29 July 1985.

3. National Bureau of Standards, Report ICST/HLNP-81-19, Security in Higher Level Protocols: Approaches, Alternatives, and Recommendations, by V. Voydock, and S. Kent, 1981.

4. Voydock, V., and Kent, S., "Security in High-Level Network Protocols," Computing Surveys, v. 15, no. 2, pp 135-171, June 1983.

5. Tanenbaum, A.S., Computer Networks, Prentice-Hall, Inc., 1981.

6. Boebert, E., Kain, R., and Young, B., "Trojan Horse Rolls Up to DP Gate," Computerworld, pp.65-69, 2 December 1985.

7. MIT Laboratory for Computer Science, Cambridge, Mass., Report LCS-TR-162, Encryption-Based Protection Protocols for Interactive User-Computer Communications, by S. Kent, 1976.

8. Diffie, W., and M.E. Hellman, "New Directions in Cryptology," IEEE Transactions on Information Theory, IT-22, pp. 644-654, 6 November 1976.

9. National Bureau of Standards, Federal Information Processing Standard, FIPS publication 46, Data Encryption Standard, January 1977.

10. Davio, M., and others, "Analytical Characteristics of the DES," Advances in Cryptology- Proceedings of Crypto 83, by D. Chaum, pp. 171-200, Plenum Press, Inc., 1983.

11. National Bureau of Standards, Federal Information Processing Standard, FIPS Publication 81, DES Modes of Operation, 2 December 1980.

12. Spencer, M.E., and Tavares, S.E., "A Layered Broadcast System," Advances in Cryptology- Proceedings of Crypto 83, by D. Chaum, pp. 157-170, Plenum Press, Inc., 1983.

13. Gemini Computers Inc., Carmel, Ca., System Overview-Gemini Trusted Multiple Microcomputer Base, 11 May 1984.

14. Gemini Computers, Inc., Monterey, Ca., GEMSOS Ring 0 User's Manual for Pascal MT+86, November 1985.

15. Brewer, D.J., A Real-Time Executive for Multiple Computer Clusters, Masters Thesis, Naval Postgraduate School, Monterey California, December 1984.

16. Digital Research, Inc., CPM-86 Operating Manual, 1983.

17. Gemini Computers, Inc., Monterey, Ca., Sysgen User's Manual, September 1985.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
4.	Dr. M.L. Cotton, Code 62Cc Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
5.	Dr. Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93943	3
6.	Lt. Philip J. Corbett, USN 72 Pilgrim Rd. Concord, Massachusetts 01742	2
7.	Daniel Green, Code 20F Naval Surface Weapons Center Dahlgren, Virginia 22449	1
8.	Capt. J. Donegan, USN PMS 400B5 Naval Sea Systems Command Washington, D.C. 20362	1
9.	RCA AEGIS Data Repository RCA Corporation Government Systems Division Mail Stop 127-327 Moorestown, N.J. 08057	1
10.	Library (Code E33-05) Naval Surface Weapons Center Dahlgren, Virginia 22449	1
11.	Dr. M.J. Gralia Applied Physics Laboratory Johns Hopkins Road Laurel, Maryland 20707	1
12.	Dana Small, Code 8242 NOSC San Diego, California 92152	1