1

1.0

1.1

1.25

1.4

1.6

1.8

2.0

2.2

2.5

2.8

3.2

3.6

4.0

4.5

5.0

5.6

6.3

7.1

8.0

9.0

10.0

11.2

12.5

14.0

16.0

18.0

20.0

22.5

25.0

28.0

31.5

36.0

40.0

45.0

50.0

56.0

63.0

71.0

80.0

90.0

100.0

112.0

125.0

140.0

160.0

180.0

200.0

225.0

250.0

280.0

315.0

360.0

400.0

450.0

500.0

560.0

630.0

710.0

800.0

900.0

1000.0

1120.0

1250.0

1400.0

1600.0

1800.0

2000.0

2250.0

2500.0

2800.0

3150.0

3600.0

4000.0

4500.0

5000.0

5600.0

6300.0

7100.0

8000.0

9000.0

10000.0

11200.0

12500.0

14000.0

16000.0

18000.0

20000.0

22500.0

25000.0

28000.0

31500.0

36000.0

40000.0

45000.0

50000.0

56000.0

63000.0

71000.0

80000.0

90000.0

100000.0

112000.0

125000.0

140000.0

160000.0

180000.0

200000.0

225000.0

250000.0

280000.0

315000.0

360000.0

400000.0

450000.0

500000.0

560000.0

630000.0

710000.0

800000.0

900000.0

1000000.0

1120000.0

1250000.0

1400000.0

1600000.0

1800000.0

2000000.0

2250000.0

2500000.0

2800000.0

3150000.0

3600000.0

4000000.0

4500000.0

5000000.0

5600000.0

6300000.0

7100000.0

8000000.0

9000000.0

10000000.0

11200000.0

12500000.0

14000000.0

16000000.0

18000000.0

20000000.0

22500000.0

25000000.0

28000000.0

31500000.0

36000000.0

40000000.0

45000000.0

50000000.0

56000000.0

63000000.0

71000000.0

80000000.0

90000000.0

100000000.0

112000000.0

125000000.0

140000000.0

160000000.0

180000000.0

200000000.0

225000000.0

250000000.0

280000000.0

315000000.0

360000000.0

400000000.0

450000000.0

500000000.0

560000000.0

630000000.0

710000000.0

800000000.0

900000000.0

1000000000.0

1120000000.0

1250000000.0

1400000000.0

1600000000.0

1800000000.0

2000000000.0

2250000000.0

2500000000.0

2800000000.0

3150000000.0

3600000000.0

4000000000.0

4500000000.0

5000000000.0

5600000000.0

6300000000.0

7100000000.0

8000000000.0

9000000000.0

10000000000.0

11200000000.0

12500000000.0

14000000000.0

16000000000.0

18000000000.0

20000000000.0

22500000000.0

25000000000.0

28000000000.0

31500000000.0

36000000000.0

40000000000.0

45000000000.0

50000000000.0

56000000000.0

63000000000.0

71000000000.0

80000000000.0

90000000000.0

100000000000.0

112000000000.0

125000000000.0

140000000000.0

160000000000.0

180000000000.0

200000000000.0

225000000000.0

250000000000.0

280000000000.0

315000000000.0

360000000000.0

400000000000.0

450000000000.0

500000000000.0

560000000000.0

630000000000.0

710000000000.0

800000000000.0

900000000000.0

1000000000000.0

1120000000000.0

1250000000000.0

1400000000000.0

1600000000000.0

1800000000000.0

2000000000000.0

2250000000000.0

2500000000000.0

2800000000000.0

3150000000000.0

3600000000000.0

4000000000000.0

4500000000000.0

5000000000000.0

5600000000000.0

6300000000000.0

7100000000000.0

8000000000000.0

9000000000000.0

10000000000000.0

11200000000000.0

12500000000000.0

14000000000000.0

16000000000000.0

18000000000000.0

20000000000000.0

22500000000000.0

25000000000000.0

28000000000000.0

31500000000

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**

National Library
of Canada

Bibliothèque nationale
du Canada

CANADIAN THESES
ON MICROFICHE

THÈSES CANADIENNES
SUR MICROFICHE

0-315-22173-9

NAME OF AUTHOR/NOM DE L'AUTEUR Robert Verrett

TITLE OF THESIS/TITRE DE LA THÈSE Design of a Secure Front End System

UNIVERSITY/UNIVERSITÉ Carleton University

DEGREE FOR WHICH THESIS WAS PRESENTED/
GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE M.Sc.

YEAR THIS DEGREE CONFERRED/ANNÉE D'OBTENTION DE CE GRADE 1987

NAME OF SUPERVISOR/NOM DU DIRECTEUR DE THÈSE Dr R. Buhr

Permission is hereby granted to the NATIONAL LIBRARY OF
CANADA to microfilm this thesis and to lend or sell copies
of the film.

The author reserves other publication rights, and neither the
thesis nor extensive extracts from it may be printed or other-
wise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈ-
QUE NATIONALE DU CANADA de microfilmer cette thèse et
de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la
thèse ni de longs extraits de celle-ci ne doivent être imprimés
ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/DATE 27 Apr 84 SIGNED/SIGNÉ Robert Verrett

PERMANENT ADDRESS/RÉSIDENCE FIXE 12 Erin Cres.
Ottawa Ont.
K1V 9Z2

DESIGN OF A SECURE
FRONT END SYSTEM

by

© ROBERT WAYNE VERRETT

A thesis submitted to the Information and Science
Committee in partial fulfillment of the requirements for
the degree of Master of Science.

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
Ottawa, Ontario, Canada

April 1984

copyright © Robert Wayne Verrett, 1984

The undersigned, recommends to the Faculty of
Graduate Studies and Research, acceptance of the thesis

" DESIGN OF A SECURE FRONT END SYSTEM "

by Robert W. Verrett, in partial fulfillment of the
requirements for the degree of Master of Science.

Thesis Supervisor
Chairman,
Department of Systems and
Computer Engineering

Department of Systems and Computer Engineering

April 1984

ACKNOWLEDGEMENTS

I wish to thank Professor Ray Buhr for his suggestions and guidance during the writing of this thesis. The interest of Milan Kuchta in this work, as well as helpful discussions, is greatly appreciated. Finally I wish to thank my wife June for her encouragement and editorial assistance.

ABSTRACT

An area of growing concern is that of computer security in communication systems. The few methodologies that are presently being used for developing secure systems are expensive in terms of time and computer resources. This thesis describes a case study of the design of a secure communication front end system using a graphic design method. A data flow graph, model graphs, detailed structure graphs, and pseudo code are presented. The aim of using a graphic design method is to contribute to the development of a simple and effective methodology for the design of secure systems. Though applied very early in the design, an evaluation criteria developed by the U.S. Department of Defense indicated that the front end system should provide labeled security protection. The significance of this thesis is in completing a case study and indicating future work in the areas of design methodologies, computer-aided design, and implementation issues.

TABLE OF CONTENTS

- i. Abstract
- ii. Table Of Contents
- iii. List Of Figures
- iv. Definitions of Terms

I. INTRODUCTION

A. Background of the Thesis

- 1. General
- 2. Why Secure Front End Systems are Needed
- 3. Major Issues

B. Description of the Thesis

- 1. Scope
- 2. System Requirements and Assumptions
- 3. Protocols
- 4. Objectives and Tools

C. Outline of the Thesis

II. SECURE SYSTEMS: COMMUNICATIONS, COMPUTERS, SOFTWARE, AND HARDWARE

A. Background On Communication Security

- 1. General
- 2. Problems Addressed
- 3. Cryptography
- 4. Tempest

B. Background On Computer Security

- 1. General
- 2. Security Objectives

- 3. Security Rules
- 4. Basic Security Requirements
- 5. Methods Of Attack
- C. Security Software
 - 1. ADA
 - 2. Euclid
 - 3. Other Computer Security Languages
- D. Issues Regarding ADA and Euclid
 - 1. General
 - 2. Advantages of CE over ADA
 - 3. Advantages of ADA over CE
 - 4. Problems With ADA
 - 5. Specification and Implementation Issues
- E. Other Definitions And Criteria
 - 1. The Security Model
 - 2. Definition of a Reference Monitor
 - 3. Definition of Trusted and Not Trusted Software
 - 4. Criteria for Trusted Software
- F. Hardware Issues
- G. Security Issues and Solutions

III. DESIGN OF THE FRONT END

- A. General
 - 1. Chapter Introduction
 - 2. Design Introduction
- B. Data Structures and Data Flow Graph
- C. Structure Graphs

1. Overall Structure Graph
2. Interface Structure Graphs
3. Manager Structure Graphs
- D. Implications of the Security Model

IV. THE SECURITY KERNEL

- A. General
- B. Security Kernel Characteristics of the Front End
 1. Requirements
 2. Restrictions Created by the Security Kernel
- C. Design of the Security Kernel
 1. Hardware - Software Tradeoffs
 2. Structure Graphs and Pseudo Code

V. THE REMAINING FRONT END SOFTWARE

- A. General
- B. The Trusted and Untrusted Software
 1. Specification of the Remaining Software
 2. Decision on Which Software is Trusted
- C. Design Of The Trusted And Not Trusted Software
 1. Trusted Software: Structure Graphs and Pseudo Code
 2. Not Trusted Software: Structure Graphs and Pseudo Code

VI. EVALUATION OF THE SYSTEM DESIGN

- A. Background On Evaluation
 1. General
 2. Items to Look For

B. The Trusted Computer Systems Evaluation Criteria

C. Application of the Criteria to the Design in
Chapters III - IV }

1. Overview

2. In Depth

VII. CONCLUSIONS AND FUTURE WORK

A. Conclusions

B. Direction of Future Work

Appendix A. Brief Description of the Design Method

Bibliography

LIST OF FIGURES

Figure I.A.1 External View of the Front End System

Figure I.A.2 Example System

Figure I.A.3 Send-Receive Matrix

Figure I.B.1 Protocol Mapping

Figure II.E.1 Classical Reference Monitor

Figure II.E.2 Ring View Of The Software

Figure III.A.1 Example : Two Layers of a Transport Model

Figure III.B.1 Data Flow Graph of the System

Figure III.C.1 Overall Structure Graph

Figure III.C.2 System Layered Structure Graph

Figure III.C.3 Terminal Interface Structure Graph

Figure III.C.4 Console Interface Structure Graph

Figure III.C.5 Network Interface Structure Graph

Figure III.C.6 Host Interface Structure Graph

Figure III.C.7 Partial Manager Structure Graph

Figure III.C.8 Hardware Solution

Figure III.C.9 Software Solution

Figure IV.B.1 System Layers and Security Requirements

Figure IV.C.1 Audit Package Structure Graph and Pseudo Code

Figure IV.C.2 Marking Package Structure Graph and Pseudo Code

Figure IV.C.3 Identification Package Structure Graph and
Pseudo Code

Figure IV.C.4 Policy Package Structure Graph and Pseudo Code

Figure V.C.1 Packet Buffer Pool

Figure V.C.2 Terminal Driver

Figure V.C.3.A Host.Driver

Figure V.C.3.B Host.Split and Host.Compress

Figure V.C.4.A Manager

Figure V.C.4.B Manager.Q-Manager

Figure V.C.4.C Manager.Command-Processor

Figure V.C.5 Generic Graph and Code For Transport Task

Figure V.C.6 Divider

Figure VI.B.1 Divisions and Classes

DEFINITION OF TERMS

- COVERT CHANNEL:** A communication channel that allows a process to transfer information in a manner that violates the system's security policy.
- DEDICATED SYSTEM:** A system in which all users are at the same classification level and there are no need-to-know restrictions.
- DISCRETIONARY ACCESS CONTROL:** A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject.
- FORMAL VERIFICATION:** The process of using formal proofs to demonstrate the consistency (design verification) between a formal specification of a system and a formal security policy model or (implementation verification) between the formal specification and its program implementation.
- MANDATORY ACCESS CONTROL:** A means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity.
- MULTILEVEL SYSTEM:** A system in which the computer itself must distinguish the levels of information sensitivity and the user authorization.

OBJECT: A passive entity that contains or receives information. Access to an object potentially implies access to the information it contains.

REFERENCE MONITOR

CONCEPT: An access control concept that refers to an abstract machine that mediates all accesses to objects by subjects.

SECURITY KERNEL: The hardware and software elements of a system that implement the reference monitor concept.

SECURITY LEVEL: The combination of a hierarchical classification and a set of nonhierarchical categories that represent the sensitivity of information.

SECURITY POLICY: The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information.

SPOOFING: The insertion of nonauthorized packets into the data stream.

SUBJECT: An active entity, generally in the form of a person, process, or device that causes information to flow among objects or changes the system state.

SYSTEM HIGH: A system in which all users are at the same classification level but not necessarily the same need-to-know.

TEMPEST: The unclassified name referring to the investigation and study of compromising emanations commonly known as radio frequency interference (RFI).

TRAP DOOR: A hidden software or hardware mechanism that permits system protection mechanisms to be circumvented. It is activated in some non-apparent manner.

TROJAN HORSE: A computer program with an apparently or actually useful function that contains additional (hidden) functions that surreptitiously exploit the legitimate authorizations of the invoking process to the detriment of security. For example, making a "blind copy" of a sensitive file for the creator of the Trojan Horse.

I INTRODUCTION

I.A BACKGROUND OF THE THESIS

I.A.1 GENERAL

The purpose of this thesis is to design and evaluate the high level specifications of a computer communications secure front end system. Figure I.A.1 shows the external view of the front end system which is connected to a host, a network, packet terminals, and a console. These are defined in section I.B.2.

Figure I.A.1 External View of the Front End System

I.A.2 WHY SECURE FRONT END SYSTEMS ARE NEEDED

A host computer can be directly connected to the host's terminals and other networks, but this would force the host to perform all of the I/O tasks required by the system. A second option is to offload these tasks onto a front end computer. This allows the host more time for processing information. If the host is processing sensitive or classified information then the additional burden of checking the security requirements can also be offloaded. This is discussed further in section I.B.3. Some of the other functions a front end can handle are: protocol conversion, concentration of incoming data, separation of outgoing data, and collection of statistics.

An example of where a secure front end could be used is illustrated in Figure I.A.2. In this example, components of the system are separated into four security domains. The Console and Secure Front End are contained in a multilevel domain; the network is in a secret domain; two terminals are represented: one top secret and the other unclassified; and finally the host, which can handle two security levels: secret and top secret. This creates a matrix like that in Figure I.A.3 of senders and receivers. For example, terminal 1 cannot send packets to the network, and in fact can only send or receive unclassified packets to or from the Console.

The matrix of security rules must be enforced by some component of the system to guarantee the integrity of the system. The front end is an ideal place to enforce these security rules. It is a crossroad for all of the communication paths and can be

better protected than any of the other components. The terminals and network cannot be physically protected easily because of user accessibility and host computers are usually vulnerable owing to software that cannot be trusted.

Figure I.A.2 Example System

Entries are the possible traffic classifications between senders and receivers:

SENDER *****	RECEIVER *****				
	Host	Network	Console	Terminal 1	Terminal 2
Host		Secret	Secret + Top Secret		Top Secret
Network	Secret		Secret		
Console	Secret + Top Sec.	Secret		Unclassif- ied	Top Secret
Terminal 1			Unclassif- ied		
Terminal 2	Top Secret		Top Secret		

Figure I.A.3 Send-Receive Matrix

I.A.3 MAJOR ISSUES

The three major issues in the design of a secure front end system are the following:

- a. Communication software that can somehow be checked to verify that it performs all the specified functions and no other hidden functions.
- b. Hardware that can support the software by protecting key portions of it from modification and ensure that sections of the software cannot be bypassed.
- c. Protection of both the hardware and the software from unauthorized changes during the full life cycle of the system.

I.B DESCRIPTION OF THE THESIS

I.B.1 SCOPE

This thesis includes the first two stages of the life cycle of a secure front end system. These stages are the requirements and the design. The stages not dealt with are: coding, integration, testing, and operation-maintenance. Only the major features of the system are addressed by the requirements and the design.

The requirements are informally specified using text and diagrams. No attempt is made to formally specify the system or verify that the requirements are fully addressed by the design.

In the design, data flow graphs, structure graphs, and pseudo code are used. The main constructs used are Ada-like tasks and packages.

I.B.2 SYSTEM REQUIREMENTS AND ASSUMPTIONS

As indicated in section I.A.1 there are four types of communication interfaces to the front end. These are a host interface, a network interface, some packet terminals, and a console.

The host interface consists of a single, high-speed, synchronous, bidirectional channel. The host protocol is assumed to require large buffers capable of holding multiple packets. Within the front end the host driver controls a sufficient number of these buffers to prevent deadlock.

The network interface is a medium speed, bidirectional packet-switching link to a network. This link is used by both

the terminals and the host to send and receive packets to and from other systems attached to the network.

There are "n" terminals connected to the front end. These are low speed packet-switching terminals which are used to prepare, send, and receive messages. The terminals must be capable of handling all the protocol of the network. Each terminal is connected to the front end via a dedicated line.

The console is a specially designated terminal which has greater capabilities than a regular terminal. The console is used to control the front end system and must always be available when the system is in use. Normally only the security officer has access to the console and the console must be protected to the highest security level of the system.

The system will be capable of performing the following functions (providing that the security requirements are abided by):

- a. Switch a packet from any communication line to any other communication line.
- b. Switch the host, network, or terminal interfaces "on" or "off".
- c. Audit all packets traveling through the system using an audit data base.
- d. Accept commands to the front end from either the host, network, or console (the commands are defined below).
- e. Dump emergency and error messages to the console.
- f. Send security violations to the console.

The front end is designed so that new commands can be easily added in the future. The original commands are as follows:

- a. From the host:
 - request statistical information,
 - connect or disconnect the network interface.
- b. From the network interface:
 - send or request network configuration information.
 - disconnect the network interface.
- c. From the console:
 - turn terminals "on" or "off",
 - connect or disconnect the host,
 - request audits and statistics from the front end system,
 - reconfigure the terminals.

All the communication lines are assumed to be secure to the level required by the particular system that the front end is being used in. Also, it is assumed that encryption and physical security is used to protect against active penetration attacks.

I.B.3 PROTOCOLS

Normally protocols are designed to handle all of the error conditions in a communication system. This is not always the case in a multilevel system. System components that are at a security level lower than the maximum of the system cannot be trusted to handle security violations. An example of this type

of system was given in section I.A.2. Hosts usually allow access to a wide variety of software and must handle many types of requests. This makes it very difficult to certify a host to be a multilevel system. On the other hand, the front end's activities are limited and well defined, making it much easier to certify. Therefore, it makes sense to use the front end to selectively audit relevant security fields in the upper and intermediate layers of the protocol. The reason the front end should not handle all the protocols for the host and terminals is that this would unnecessarily complicate the front end and increase the work needed for certification.

It is assumed that the front end is in a homogeneous network and that the protocol messages include at least the following:

- the sender's address,
- the receiver's address,
- the classification of the packet.

If any of these are not included they will have to be incorporated into the protocols. The layer at which they will be added depends on the particular protocols and network architecture used. An attempt has been made not to restrict the application of the front end to a particular protocol model. The front end should appear transparent in any authorized connection and should terminate unauthorized connections without the end parties knowing why.

Figure I.B.1 shows a mapping of a partitioning of the system protocols. The secure front end handles low-level proto-

cols for the local channels between itself, the host, the terminals, and the console. It also handles the network interface. The host, terminals, and console must handle the intermediate- and upper-level protocols, which are selectively audited by the front end.

Figure I.B.1 Protocol Mapping

I.B.4 OBJECTIVES AND TOOLS

The objectives of this thesis are the following:

- a. To perform a case study on the design of a secure front end system with an aim of contributing to the clarification of security issues in computers and networks.
- b. To perform a case study of the application of the graphical design methodology " System Design With ADA " [BUH84] by using it to design the front end system, with the aim of contributing to the development of a methodology for the design of secure systems.
- c. To contribute to an understanding of how to apply the Trusted Computer System Evaluation Criteria [CSC83] to designs developed using future methodologies arising from this work.

I.C OUTLINE OF THE THESIS

Chapter I is the introduction to the thesis. It includes the reasons for secure front end systems and the major issues in secure front ends. The scope, the assumptions, and the expected results of the thesis are also discussed.

The second chapter deals with the general background of secure systems and covers communications, computers, software, and hardware. In communication security the problems are addressed along with the solutions of cryptography and tempest. Computer security is divided into three areas: general discussion, software, hardware. Under general discussion the security objectives, security rules, and basic requirements are discussed. Also discussed are storage and timing channels, and methods of attacking computer systems. ADA, Euclid, and other computer security languages are dealt with in the software section. Section II.E deals with the security model, reference model, trusted and not trusted software, and the criteria for trusted software. The chapter is rounded off by a discussion of the hardware issues in secure systems and some of the security issues and solutions.

Chapter III is on the overall design of a secure front end and presents the canonical forms of the structure graphs. There are also an introduction of the steps taken in the design, and a discussion of the implications of the security model. Section III.B defines the data structures and presents a data flow graph.

Chapter IV discusses the security kernel. Character-

istics of a security kernel on a front end system are discussed, including the requirements and restrictions. Finally the security kernel is designed using structure graphs and pseudo code.

The fifth chapter defines the remaining front end software. It is classed as being trusted or not trusted and designs using structure graphs and pseudo code are included for the major components.

Chapter VI is an evaluation of the full system. Also included is a brief history of evaluation techniques and the reasons for choosing the "Trusted Computer System Evaluation Criteria" [CSC83] for the evaluation.

Chapter VII contains conclusions on the design methodology and the evaluation technique. Recommendations on embedding Euclid in ADA software and on systems using ADA are also in this chapter.

The thesis is rounded out with a brief description of the design method in Appendix A and a Bibliography.

II SECURE SYSTEMS: COMMUNICATIONS, COMPUTERS, SOFTWARE,
 AND HARDWARE

II.A BACKGROUND ON COMMUNICATION SECURITY

II.A.1 GENERAL

The introduction of powerful, low-cost computers and the ever increasing sophistication of telecommunications networks has created an environment in which voice and data communications are becoming more susceptible to interception [NYE81]. The age of electronics has provided the ability to intercept communications of almost any form. For a few hundred dollars a system can be built to tap a telephone line; less than five thousand dollars can buy equipment to intercept microwave signals; and for under twenty-five thousand dollars satellite transmissions can be intercepted (e.g. dishes used to receive Pay-TV signals from the U.S.A. can be purchased for under five thousand dollars). The possible interceptors range from hobbyists and experimenters to thrill seekers and professional attackers.

The attempts to counter the interception of information is called Communication Security. Communication Security is concerned with the protection of sensitive and classified communications over any form of communication channel. This includes both digital and analog transmissions. The Canadian government currently classifies information into five levels of security. From lowest to highest the levels are: Unclassified, Restricted, Confidential, Secret, and Top Secret. Each level has tighter security handling restrictions than the levels below it.

II.A.2 PROBLEMS ADDRESSED

Some of the problems addressed by communication security are: authentication, authorization, access control, privacy, and data security. Authentication deals with positive identification of the communicating parties and ensuring that some form of spoofing is not taking place. Authorization and access control are both involved with who is authorized to read, create, write, or send certain information. An example is authorization codes used by banks to transfer money. Privacy deals with the protection of services such as scrambling PAY-TV signals. Data security deals with the protection of information from disclosure.

II.A.3 CRYPTOGRAPHY

One of the main tools in communication security is cryptography. Cryptography is a method of transforming information to hide the contents of the original information. The transformations range from simple substitution ciphers where each letter of the alphabet is substituted for another letter to secret algorithms used by governments to secure classified government communications.

For analog communications there are two general techniques used for encryption:

1. Scrambling of the analog signals,
2. Converting the analog signal to digital form, using a digital encryption technique, and reconverting to analog at the other end.

For digital communications there are two main encryption

methods:

1. Block encryption - this method divides the plaintext into fixed blocks and encrypts each block independently.
2. Stream encryption - in this method the cryptographic equipment generates a continuous random data stream (called the keystream) which is added to the plaintext to yield ciphertext.

Stream encryption offers an extra advantage in that if the plaintext is not being sent then the keystream alone can be transmitted. Because the keystream is not distinguishable from ciphertext anyone intercepting the traffic cannot tell when actual messages are being transmitted. This is known as Traffic Flow Security [NYE81].

The Data Encryption Standard (DES) is a good example of a cryptographic algorithm. A full description can be found in FIPS Publication 46 [FIP77]. DES uses a secret 64 bit key (56 random bits and 8 check bits). The security of DES rests solely on this key since the algorithm is public knowledge. DES can be used for block encryption and stream encryption.

II.A.4 TEMPEST

One area of concern for communication security is tempest. Tempest is an unclassified name referring to the investigation and study of compromising emanations or leaky electronic pulses commonly known as radio frequency interference (RFI). These emanations can be intercepted, analyzed, and the infor-

mation extracted if equipment is not properly shielded. The U.S. Government has established a standard called NACSEM 5100 to limit the amount of RFI leakage that is acceptable. The actual specifications within NACSEM 5100 are classified [NYE81].

II.B BACKGROUND ON COMPUTER SECURITY

II.B.1 GENERAL

Computer security concerns the protection of computers and their services from all natural and man-made hazards. The problem has increased as the complexity and the interconnection of computers has evolved. In a dedicated single user mode, the protection is procedural in that the user must remember to clear the memory and the temporary storage after completing a task. As systems increase in complexity and more users are added, the security problem becomes more acute. Protection of data and programs between users becomes a concern and involves physical, personnel, hardware, software, and operational procedures. Protection is needed to [TUR81]:

1. Safeguard assets and resources,
2. Comply with laws and regulations,
3. Enforce management control,
4. Assure the safety and integrity of computer controlled processes and systems.

Three concepts of a computer system have evolved. They are: dedicated, system high, and multilevel. Dedicated systems are those in which all users are at the same classification level and there are no need-to-know restrictions. In a system high mode, the use of a system is restricted to users at the same classification level but not necessarily the same need-to-know. The security protection in both dedicated and system high computer systems are derived from physical security, personnel security, and the use of secure communications. In a multilevel mode, the com-

puter itself must distinguish between the levels of information sensitivity and the user authorization.

II.B.2 SECURITY OBJECTIVES

Three security objectives for computer systems have been defined [FIP80]. They are:

1. Data Confidentiality - to ensure the protection of data from accidental or intentional disclosure to unauthorized users.
2. Data Integrity - to ensure the protection of data against accidental or malicious destruction, or alteration by unauthorized users.
3. Availability of Service - to safeguard against malicious denial of service to authorized users.

II.B.3 SECURITY RULES

Documents are each given a classification Level "L" and a set of categories "C" (possibly empty). The levels "L" are strictly ordered:

Top Secret > Secret > Confidential > Unclassified

The set of categories "C" tend to have no ordering.

The two fundamental security rules are [IEE82]:

1. An individual can only have read access to a document if the individual's clearance level is greater than or equal to the document's classification level and if the individual is cleared for all the categories applied to the document.
2. Only a specially authorized individual may reduce

the level or remove categories from a classified document.

II.B.4 BASIC SECURITY REQUIREMENTS

In [CSC83], six basic requirements are defined that should be met by a computer system handling or processing classified information. They are: security policy, marking, identification, accountability, assurance, and continuous protection. Depending on the success of meeting the above six requirements, [CSC83] defines an evaluation criteria for rating the computer system. This is discussed further in Chapter VI.

II.B.5 METHODS OF ATTACK

Three methods used to attack a computer system are [MYE80]: accidental disclosure, penetration, and subversion. Accidental disclosure relies on a probabilistic event that the attacker cannot control. This includes items such as mounting the wrong disk or tape, hardware failure, or carelessness of an operator. Penetration attacks deal with deliberate attempts to circumvent the system controls. Some good examples are the U.S. tiger team efforts set up in the 70's [SCH79]. Subversion is a covert attack on the controls of a system to allow undetected access to information. Three steps occur in a subversive attack:

1. the insertion of code (often called a Trojan horse or a Trap Door);
2. the exercising of this code; and
3. the retrieval of the resulting information.

Storage and timing channels are methods of transferring information between individuals. These channels are classed as either overt or covert. Overt channels are structures that are normally associated with a computer such as buffers, files, and I/O devices. Overt channels can be detected and controlled by enforcing the security requirements. Covert channels are entities not normally considered as structures to transfer information. These include the opening and closing of files, the setting and resetting of flags or file locks, and the passing of time between events. Covert channels are much more elusive to find and control.

II.C SECURITY SOFTWARE

II.C.1 ADA

In 1975 the U.S. Government formed the Department of Defence (DOD) Common High Order Language Program to establish a single high order computer programming language for the DOD embedded computer system. The requirements were refined as a result of wide and public consultation, are reported in documents entitled STRAWMAN, WOODENMAN, TINMAN, IRONMAN, STEELMAN, and METHODMAN. TINMAN was an evaluation of existing languages and resulted in four major conclusions [BAR80]:

1. No language was suitable in its present form.
2. A single language was a desirable goal.
3. The state-of-the-art can meet the requirements.
4. Development should start from a suitable base.

Pascal [JEN 75] was used as the base by all four of the final contractors. The final language chosen was defined by CII Honeywell Bull. ADA was picked by DOD as the name of the new language.

The main modern features of ADA are:

- Strong typing.
- Mechanisms for encapsulation and separate compilation.
- Exception handling.
- Tasks and Packages.
- High Level concurrency using Rendez-vous.

ADA is designed as a common programming language for an assortment of applications including large scale and real-time systems.

II.C.2 EUCLID

Euclid was designed in 1975 as a programming language for the development of verifiable systems. Full Euclid is described in [LAM77]. In 1976 a subset called Toronto Euclid was implemented by the University of Toronto and I.P. Sharp Associates under contracts with the Canadian Department of National Defence and the U.S. Department of Defense [IPS80]. Further research lead to the design and implementation of the language Concurrent Euclid (CE) at the University of Toronto in 1981 [COR81] and [HOL81], and Ottawa Euclid (OE) by I.P. Sharp Associates [CRO82].

Euclid, like ADA, is based on Pascal [JEN75]. CE is a subset of Euclid to which a set of concurrency extensions based on monitors [HOA74] has been added.

The major extensions that CE added to Pascal are the following [HOL81]:

1. Separate compilation.
2. Modules - packages of data and procedures/functions that access the data.
3. Concurrency including monitors, processes, signal, wait, and busy.
4. Control of scope (discussed in section II.D).
5. System programming constructs such as variables at absolute addresses.

The Pascal features that CE does not support include floating point, enumerated types, and procedures or functions nested inside other procedures or functions.

II.C.3 OTHER COMPUTER SECURITY LANGUAGES

There exist many other computer languages and methodologies developed with verification in mind. A sample of these are:

1. Formal Development Methodology (FDM) by System Development Corporation. FDM uses Ina Jo as the specification language and also includes an Interactive Theorem Prover and a Verification Condition Generator [CSI80].
2. Gypsy by the University Of Texas. A Pascal-based language which can be used for both formal specification and implementation [CSI80].
3. Hierarchical Development Methodology (HDM) by Stanford Research Institute. HDM is an integrated collection of a language (SPECIAL), tools, concepts, and guidelines [NBS80].

II.D ISSUES REGARDING ADA AND EUCLID

II.D.1 GENERAL

ADA is a powerful and complex language that requires an extensive run-time support package. These facts make ADA very difficult if not impossible to verify. The design of ADA has been criticized in articles and lectures, i.e. [ADA81] and [VAN80]. Euclid on the other hand was specifically designed with verification in mind. Concurrent Euclid (CE) programs can be run on a bare machine and require only a small assembly language kernel. Because ADA allows the insertion of low-level code, CE programs can be embedded in ADA software when highly reliable software is needed.

II.D.2 ADVANTAGES OF CE OVER ADA

Some of the advantages of CE over ADA are:

- a. CE exhibits tighter control on the scope of variables, types, and subprograms than ADA. In CE, import and export lists are used to define the scope of all names. In ADA, names can be automatically inherited by scope.
- b. CE has a small run-time support package versus ADA's extensive run-time support package.
- c. The monitor constructs in CE give more control over priority and scheduling than the rendez-vous mechanism in ADA.
- d. CE has constructs specifically for verification. These are assert, pre-and post-condition checking.

II.D.3 ADVANTAGES OF ADA OVER CE

Some of the advantages of ADA over CE are:

- a. Because ADA's rendez-vous is a higher level coordination mechanism than CE's signal and wait, it should be easier to verify task interactions in ADA.
- b. ADA supports floating point. This is not supported in CE.

II.D.4 PROBLEMS WITH ADA

Some of the possible problem areas in ADA are:

- a. ADA allows overloading which is not desirable from a verification point of view [INT81].
- b. Exceptions are extremely undesirable in terms of verifiability because they allow a task to leave its scope entirely and an exception can propagate several levels [INT81].
- c. Time-outs in the rendez-vous mechanism can cause deadlock. This can occur if a task accepts a call that has just timed-out.
- d. The ADA rendez-vous can be restrictive. An example is a resource scheduler that requires a dispatcher.
- e. ADA [INT81] appears to have a lack of uniformity in parameter passing during a rendez-vous:
 - in uniprocessors: parameters can still be changed by other processes when the calling task is blocked.
 - in multiprocessors: a rendez-vous may cross pro-

cessor boundaries, parameters are sent in the form of messages and therefore are immune to further changes.

II.D.5 SPECIFICATION AND IMPLEMENTATION ISSUES

The front end will be specified using the ADA graphical design method described in [BUH84] and briefly summarized in Appendix A. The reasons for choosing this method are the following :

1. The clarity of organization achieved by using graphics.
2. The potential implementation of ADA programs in hardware [CAR84].
3. The possible use of computer-aided design systems such as the CADA (Computer-Aided Design with Ada) project at Carleton University, Ottawa.

Though both ADA and CE were heavily influenced by the Pascal programming language, their respective approaches to concurrency are very different. The rendez-vous mechanism in ADA requires little knowledge of the principles of concurrency in order to use it. On the other hand, the "wait", "signal", and "monitor" constructs in CE require a knowledge of concurrency. The ADA task resembles a "monitor" if it does not make any rendez-vous calls that could be blocked. This leads to the use of transport tasks to pass information to and from major tasks and to do the waiting for these major tasks. This is called the Transport model and an example is given in section III.A.2.

The three choices for coding the software for the front end are the following:

- a. all in ADA,
- b. all in Concurrent Euclid,
- c. an ADA shell with internal sequential Euclid code.

Problems can arise in case a. because of the security problems with ADA (see section II.D.4.). In case b. the concurrency mechanisms used by CE have not been verified and this would be very difficult because of their low level. This leaves case c., which represents a balance of high level concurrency using ADA tasks and verifiable sequential code segments.

II.E OTHER DEFINITIONS AND CRITERIA

II.E.1 THE SECURITY MODEL

A security model is the specification of the way in which the security policy will be enforced by the computer system. The basic security requirements discussed in section II.B.4 are addressed by this model. Formal specification and verification of the security model is possible [NBS80] but will not be attempted since these methodologies do not lend themselves easily to high-level coding in ADA or Euclid.

The security model describes exactly what security means in a system. Article [LAN81] describes a wide selection of formal models. These include finite state machine models, information flow models, and access matrix models. One noteworthy model is the Bell and LaPadula model. This model defines two main properties for a state to be secure, the simple security property (no subject has read access to any object of higher level) and the *-property (a subject cannot write an object to a lower level).

The following security model defines to what degree the six basic requirements that are specified in [CSC83] are to be enforced in the secure front end. Also included are the other criteria that must be enforced. For this design the six basic requirements are :

1. A security policy must enforce mandatory security as follows:
 - any line of higher classification cannot send packets to a line of lower classification (A line

is defined as either the host channel, the network interface, a terminal, or the console).

- the host and console are the only devices that can change the classification of a terminal under the host's control. A terminal's classification can only be changed up to the classification of the host.

2. There must be an access control label on all packets handled by the system. This label must reliably identify the packet's classification level and security categories.
3. Every packet must contain an identification of the sender and the receiver. The identification must be checked against the packet's markings to verify the correct access requirements.
4. An audit record must be made whenever a packet is received by the front end and the record entry must be checked and deleted when the packet leaves the front end.
5. The front end must contain hardware and software mechanisms to ensure that the system enforces requirements 1 through 4.
6. The trusted mechanisms in the front end system must be continuously protected against tampering or unauthorized changes throughout the system's life-cycle.

II.E.2 DEFINITION OF A REFERENCE MONITOR

A reference monitor is a concept or abstract machine that enforces the security policy and the access relationships between subjects and objects in a system. Subjects include users and user initiated programs, whereas objects include files, data, and peripherals.

There are three properties that are enforced by a reference monitor: completeness, isolation, and correctness. The completeness property states that the reference monitor must be invoked in all accesses from subjects to objects. The isolation property states that the monitor must be tamper-proof from both a hardware and a software point of view. These first two properties are mainly enforced by hardware mechanisms. The third property, correctness, states that the monitor should be small and well-structured so that it can be subjected to a complete security analysis.

The implementation of the reference monitor is referred to as the security kernel. A description of the security kernel for the front end system is in Chapter IV.

The classical representation of a reference monitor is illustrated in Figure II.E.1. The only access that subjects have to objects in a system is through the reference monitor. Both software and hardware are required: software is needed for flexibility in implementing the security policy and hardware is needed to enforce the monitor's properties that are not possible to enforce in software.

Figure II.E.1 Classical Reference Monitor

II.E.3 DEFINITION OF TRUSTED AND NOT TRUSTED SOFTWARE

The system software can be viewed as a ring structure with the security kernel software in the middle followed by a ring of trusted software and finally a ring of not trusted software. This is shown in Figure II.E.2.

The software, excluding the security kernel, is divided into the two categories, trusted and not trusted software. Trusted implies software that has the capability of bypassing the security model restrictions and therefore must be validated so that it only implements its specifications. On the other

hand, for not trusted software, one needs only to show that the software cannot bypass the hardware and software mechanisms set up to protect the system.

An example of trusted software is the Packet Buffer Pool (Fig. V.C.1). It must have write access to the buffers in order to clear them when they are returned to the buffer pool. The Packet Buffer Pool must be validated so that it will not overwrite buffers at any other time, thereby contaminating headers or data. An example of not trusted software are the buffers themselves. In this case the hardware and software mechanism that is set up to protect the system is the enforcement of the rule stating that the buffer software never be executed.

Figure II.E.2 Ring View Of The Software

II.E.4 CRITERIA FOR TRUSTED SOFTWARE

The criteria used to distinguish between a trusted and a not trusted software module depends upon the interaction of the module with the security kernel. If the module generates or has the capability of generating or modifying information being passed directly or indirectly to the kernel, and the kernel uses this information to make security relevant decisions then the module is classed as trusted. If the above condition is not met then the module is classed as not trusted.

II.F HARDWARE ISSUES

The hardware base can have a significant impact on the resulting system's size, complexity, and architecture [ART81]. Security features in the hardware can diminish the amount and complexity of code and simplify the verification of a system. Features not implemented in the hardware must somehow be programmed or removed from the requirements.

Some of the useful features in hardware as far as security is concerned are :

- Segmented memory to control read, write, and execute access to programs and data.
- Protection domains (such as those implemented on the Intel 286 [INT83]) to isolate the kernel and trusted software.
- Error detecting memory for reliability.
- The availability of a master-checker mode to trap intermittent hardware errors.

As mentioned in section II.E.2, hardware plays a key role in constructing a reference monitor. Both completeness and isolation are normally enforced by hardware. To make a Honeywell SCOMP processor, special boards are added to their Level 6 minicomputer. Among other things these boards monitor all bus activity. This enforces the security policy by blocking unauthorized access to memory or I/O.

II.G SECURITY ISSUES AND SOLUTIONS

This section is an informal attempt to define and discuss various rules and items that should be considered in an evaluation. These are items that are not explicitly covered by the evaluation criteria discussed in section (VI.B).

There are a number of good rules to follow when interactive multiuser terminals are attached to a system. The most obvious is that the default on all access should be the lack of access, thereby closing many possible holes in a system.

The problem of spoofing on a terminal can be foiled by a secure break key which must be used during the log on procedure to communicate directly with the operating system. This will protect against the classic spoofing attack of writing a program that emulates the log on procedure and covertly acquires other users passwords. Assuming the terminal and communication line are tamper resistant, a secure break key would communicate directly with the security kernel and therefore bypass any spoofing program.

A major problem in any multiuser system is the foolproof identification of users. There are a number of possible solutions including passwords, ID cards, handprints, question and answer schemes, and encryption. Not all solutions will work with all system and each solution has its own problems. Passwords are usually difficult to remember. ID cards can be lost or stolen. Handprints or voiceprints require sophisticated technology and are currently unreliable. Question and answer schemes rely on users entering questions and answers that are only known to them-

selves. Problems can arise when answers are forgotten.

Encryption is expensive and key distribution can be a problem.

A good rule to follow is that the design should not rely on any of the details being kept secret. The wider the distribution of the software the sooner possible flaws will be found and corrected.

There are a number of penetration weaknesses that can be addressed at the design stage of a system. Some of the penetration weaknesses are:

- a. The improper choice of the protection domain or security partition.
- b. The choice of an inappropriate initial security state.
- c. The use of nonvirtual resources such as real I/O.
- d. The improper validation of out of bounds addresses and unvalidated return arguments.
- e. Incomplete interrupt handling.
- f. Deadlock or deadly embrace conditions.
- g. Race conditions in critical sections of code.
- h. A variable that changes value between the time it is checked and the time it is used. This can be solved by using "call by value" rather than "call by reference".
- i. The improper recovery from errors or failures.

III DESIGN OF THE FRONT END

III.A GENERAL

III.A.1 CHAPTER INTRODUCTION

The purpose of this chapter is to define the overall logical structure of the secure front end. Section III.A.2 is the design introduction and defines the steps taken in the design. Section III.B presents the data structures used and a general data flow graph of the system. Section III.C is divided into three sections. III.C.1 discusses the overall structure graph. The terminal, console, network, and host interface packages are described in section III.C.2. The manager package structure graph is detailed in section III.C.3.

III.A.2 DESIGN INTRODUCTION

The steps taken in the design of the secure front end are drawn from reference [BUH84]. There are two basic steps : the data flow graphs and the structure graphs. The data flow graph shows the major data paths and is described in section III.B. The structure graphs are presented in two forms. The graphs in section III.C give the canonical or model forms and chapters IV and V give the detailed structure graphs and pseudo code.

The model, which is used for the design of the front end, is the transport model [BUH84]. It was chosen for the reason outlined in section II.D.5. A simple example of this model is given in figure III.A.1, which shows a structure graph of two layers of a system with bidirectional communications. It

has two active packages, each with a main internal task and two transport tasks used for waiting at the layer above. The main tasks never make external calls and therefore are never blocked.

Figure III.A.1 Example : Two Layers of a Transport Model

III.B DATA STRUCTURES AND DATA FLOW GRAPH

The two main data structures used for passing information are packets (p) and Host-Buffers. These are shown in Fig III.B.1. Host-Buffers are used between the host system and the host interface only. Throughout the remaining system, packets are passed. Control information is also passed between the shared modules, the interfaces, and the system manager for control and audit purposes.

The four tables shown in the System Manager are the following:

- a. An audit table containing entries made when packets enter the front end. The entries are cleared when the data is either forwarded out of the front end or processed by the front end.
- b. The statistics tables contain information on different aspects of the front end. These tables are set up to be used by the host system and console.
- c. The password tables are used to verify sign ons from the terminals.
- d. The security tables are for the use of the system manager in checking security clearances and access rights of the packets as they are routed through the front end.

Figure III.B.1 Data Flow Graph of the System

III.C STRUCTURE GRAPHS

III.C.1 OVERALL STRUCTURE GRAPH

Figure III.C.1 is the overall structure graph of the system that is derived from Figure III.B.1 using the transport model. Communication with the front end is accomplished by the four types of interface packages labeled Console, Host, Network, and Terminal. These are expanded in section III.C.2. The Manager is discussed in section III.C.3.

From Figure III.C.1 one can derive a layered or ring structure. Figure III.C.2 is an example of the layered structure with the security kernel as the inner layer.

Figure III.C.1 Overall Structure Graph

Figure III.C.2 System Layered Structure Graph

III.C.2 Interface Structure Graphs

Figures III.C.3, III.C.4, III.C.5 and III.C.6 are structure graphs representing the four main interfaces; terminals, console, network, and host respectively. All of them pass packets to and from the Manager by using transport tasks. This frees the driver tasks and Q task from the possibility of being blocked at the Manager. The Packet-Buffer-Pool is a task used by the system to handle the packet pool. For simplicity the disk will be considered as an extension of the Packet-Buffer-Pool.

The terminal interface structure graph in Figure III.C.3 includes just one of the "n" terminal drivers. The terminal packages interact directly with the Manager for forwarding packets and call a common active package Divider to pick up packets being sent to the terminal. The Divider package collects all the packets going to the terminals and internally queues them for each terminal. The terminal drivers access their packets through a common procedure Out. If no packets are waiting the terminal is internally blocked in the Divider.

Figures III.C.4 and III.C.5 are the Console and Network interface structure graphs. Both are active packages, which interface with the Manager by internal transport tasks.

The host interface structure graph is in Figure III.C.6. The driver interacts with the host using Host-Buffers. Because the Manager only handles packets, a Split task is needed to convert Host-Buffers to packets. On the send side a Compress task assembles the packets into Host-Buffers and passes the buffers to the Driver to be sent to the host.

Figure III.C.4 Console Interface Structure Graph

Figure III.C.5 Network Interface Structure Graph

Figure III.C.6 Host Interface Structure Graph

III.C.3 Manager Structure Graphs

The active package Manager (Figure III.C.7) represents the heart of the front end. The Manager contains two tasks, Q-Manager and Command-Processor, and the active package Security-Kernel. The two tasks are needed to separate the asynchronous functions of switching the traffic and responding to external queries.

The Q-Manager manages the queues for outgoing packets to the Console, Network, Host, and a common queue for all packets going to terminals. The Divider manages the queues for individual terminals (described in section III.C.2).

The Command-Processor handles all the requests for action by the front end. These include checking passwords and forwarding statistics to the Console and Host.

The internal parts of the security kernel are described in Chapter IV. The security kernel acts as a gateway for certain calls between other packages. There are different methods of trapping these calls. The first is via special reference monitor hardware which monitors the processor bus for certain addresses (Figure III.C.8). As far as software is concerned, this is the simplest method, but it can be awkward since the hardware needs to be reinitialized whenever these addresses are changed. A second approach is to use the concept of trusted software (software that has been evaluated and assured to make calls to the security kernel when required). This second approach is the one used in this design.

The two tasks in the Manager call the security kernel directly. The packages outside of Manager call entry procedures in the Manager to get to the security kernel. These calls can also be considered direct (see Figure III.C.9).

Figure III.C.8 Hardware Solution

Figure III.C.9 Software Solution

III.D IMPLICATIONS OF THE SECURITY MODEL

The design in III.C can be viewed as four layers of software: driver, transfer, management, and security kernel layers. Each of the first three layers will need to interact with the security kernel layer to address the security model defined in section II.E.1.

The driver layer includes the drivers for the host, link, and all terminals. The converter program used by the host is also included. Each of these units of software must interface with the Packet-Buffer-Pool. At this layer both (2) marking and (4) accountability must be addressed. (The numbers refer to security requirements listed in section II.E.1.)

Access is required to the audit tables and buffers.

The transfer layer consists of all the transport processes and the terminal concentrator packages. Because these units of software only handle pointers, none of the basic requirements need to be addressed.

The management layer is the manager package excluding the security kernel. The Q-Manager requires limited access to the buffers in order to obtain the intended receiver of the packet for switching purposes. The main requirements that need to be addressed are (1) security policy and (3) identification. The management layer also includes the command process and requires access to all tables and buffers in the system. (1) Security policy and (4) accountability are the requirements addressed at this layer.

The security requirements (5) assurance and (6) contin-

uous protection must be enforced throughout the system. Enforcement of requirement (6) is simplified by the fact that no user written programs are run on the front end. Therefore, the system only needs to protect itself from software that is written specifically for the system.

IV THE SECURITY KERNEL

IV.A GENERAL

The purpose of this chapter is to design the security kernel for the front end system. The principles of a security kernel are discussed in section II.E.2. The security kernel must enforce the security model as defined in section II.E.1. Section III.D deals with the implications of the security model. This includes the interaction of the different layers and the security kernel.

The requirements and restrictions of the security kernel are discussed in section IV.B. Section IV.C.1 examines the hardware and software tradeoffs of the kernel. The structure graphs and pseudo code are presented in section IV.C.2.

IV.B SECURITY KERNEL CHARACTERISTICS OF THE FRONT END

IV.B.1 REQUIREMENTS

The following refers to figure IV.B.1, which is an update of figure III.C.2. The driver layer must enforce two of the basic security requirements: marking and accountability. In the terminal, console, and link drivers these requirements can be checked immediately after a packet is received and just before sending a packet. In the host driver, on the receiving side, these checks must be delayed until the converter splits the host-buffer into packets. On the sending side the checking occurs just before the packets are compressed into host-buffers.

The management layer must enforce the security policy, accountability, and the identification requirements. These can be checked by the Q-manager task, which has access to the audit and the security tables, and the command-processor, which has access to the audit and security tables.

Figure IV.B.1 System Layered Structure Graph

IV.B.2 RESTRICTIONS CREATED BY THE SECURITY KERNEL

The security kernel creates some unconventional problems for the front end. The front end must be able to forcibly abort conversations being held between any communication lines under its control if the security rules are broken. It must also enforce some form of flow control to ensure that packets are not sent through the front end in an effort to jam the system. Some other situations that need to be studied are spoofing, the insertion of unauthorized packets into the data stream, and handling unauthorized access to the Command-Processor in the Manager.

To forcibly abort a conversation there are two basic methods. The first is to stop the receiving and sending of packets over the circuit and let normal error handling terminate the circuit as if one of the participants had aborted the circuit. This is time consuming because it requires active participation of the front end to continuously monitor the circuit to guarantee that future packets do not get sent over it. The second method is to terminate the circuit in an orderly manner. This requires the front end to generate special security termination messages to each participant.

Using flow control to prevent jamming will depend upon the speeds of each communication line and the overall capacity of the front end. Jamming can come from either the network link or a terminal. Jamming from the network can be handled by limiting the number of packets accepted per unit time. Since the terminals are under direct control of the front end, a terminal can

be switched off if it shows any unusual behavior.

Spoofing can be handled in many ways. A couple are by sequence numbers on packets and by link and end-to-end encryption.

To protect against unauthorized access to the Command-Processor, the security policy must be enforced and the Command-Processor needs to be trusted software (see Chapter V).

IV.C DESIGN OF THE SECURITY KERNEL

IV.C.1 HARDWARE-SOFTWARE TRADEOFFS

The security requirements (section II.B.4) such as continuous protection and assurance cannot be efficiently implemented in the software because of the volatile nature and speed constraints of software.

Continuous protection requires the checking of each instruction to verify that it does not modify the software or hardware mechanisms that enforce security. In software this would require a module to interrogate each system instruction before execution. This module itself would have to somehow be verified to guarantee its integrity. On the other hand a hardware module inserted between the memory and the main processor can monitor all ingoing and outgoing bus accesses.

Assurance is enforced by a combination of hardware, software, and good verification techniques. This is made easier because there is no direct user programming or access to tables in the system except by the console. The console is assumed to have access to system tables. The hardware aspect of assurance is to arbitrate access to key portions of the system. This can be done by a form of memory management as follows:

- a. check that security-related tables are never executed;
- b. enforce execute only on security software;
- c. allow execution of security software to begin only at specific entry points;

- d. limit access to security-related tables to predefined memory segments;
- e. limit access of the disk and buffers to predefined memory segments.

IV.C.2 STRUCTURE GRAPHS AND PSEUDO CODE

The software for the security kernel consists of four packages: Audit (Figure IV.C.1), Marking (Figure IV.C.2), Identification (Figure IV.C.3), and Policy (Figure IV.C.4). The only active package is Audit, which must insure mutual exclusion on the audit and statistics tables.

The line drivers call both the Audit and the Marking package. Only the In and Out entries of the Audit package can be called by these drivers. This prevents the drivers from requesting statistical information without going through the Manager. Audit.In and Marking.Check must be called in the receive module of each driver for each and every packet received. Audit.Out and Marking.Check must also be called in the send module of each driver for each and every packet sent.

The management layer consists of the Q-Manager and the Command-Processor tasks. The Identification and Policy passive packages are called by the Q-Manager to check that the sender of the packet is identified and to ensure that the security policy is enforced. From the Command-Processor the active package Audit and the passive package Policy are called. The Audit.In entry is called to acknowledge packets sent to the front end. The Audit.Out entry is called to enter information on packets initiated by

the Command-Processor. The entry Audit.Statistics might also be called if statistical information is required by the Command-Processor. The package Policy is required to check if all the outgoing packets created by the Command-Processor conform to the security policy.

Task Body Auditor is

```

...
Begin Loop
  Select
    Accept In ( P.header, Return );
    Check the header;
    Queue the request;
    set Return;
  Or Accept Out( P.header, Return );
    Check the header;
    Dequeue the request;
    set Return;
  Or Accept Stat( Request, Data );
    Check validity;
    Format statistics;
    Return in Data;
  End Select;
End Loop;
End Auditor;

```

Figure IV.C.1 Audit Package Structure Graph and Pseudo Code

Marking (passive package)

Package Body Marking;

```

...
Procedure Check( P.header, Return );
...
Begin
    Check the consistency of the marking;
    Set Return;
End Check;
End Marking;

```

Figure IV.C.2 Marking Package Structure Graph and Pseudo Code

Identification (passive package)

Package Body Identification;

```

...
Procedure Id( P.header, Return );
...
Begin
    Check the identification on the header;
    Set Return;
End Id;
End Identification;

```

Figure IV.C.3 Identification Package Structure Graph and Pseudo Code

Package Body Policy;

```

...
  Procédure Security_Check( P.header, Return );
  ...
    Begin
      Check that the Security policy is enforced;
      Set Return;
    End; Security_Check;
End Policy;

```

Figure IV.C.4 Policy Package Structure Graph and Pseudo Code

V THE REMAINING FRONT END SOFTWARE

V.A GENERAL

In section V.B the remaining front end software is specified and grouped into trusted and not trusted software. The distinction between the two is discussed in section II.E.3. Section V.C has the structure graphs and pseudo code for the trusted and not trusted software.

V.B THE TRUSTED AND NOT TRUSTED SOFTWARE

V.B.1 SPECIFICATION OF THE REMAINING SOFTWARE

The software modules that are not included in the kernel are the following:

- a. the Packet-Buffer-Pool (and the disk driver);
- b. the communication line drivers which are:
Terminal.Driver, Console.Driver,
Network-Interface.Driver, and Host;
- c. the transport tasks;
- d. the Divider.Q task;
- e. two internal tasks in Manager: Q-Manager and
Command-Processor.

This is just a skeletal design of a secure front end system. In a full design there would be other software such as performance monitors, terminal routing software, and discretionary security checking tasks which may be required to meet the specifications of the user.

V.B.2 DECISION ON WHICH SOFTWARE IS TRUSTED

The Packet-Buffer-Pool (see a. above) enforces the "activity security rule" which states that only buffers that are active can be assessed and when a buffer becomes inactive its contents must be erased. The erasure must be checked to ensure that it is complete. The buffer pool manager must also enforce the "tranquility rule" which states that the security level of active objects (which includes buffers) cannot be changed. Because of the need to enforce the above two security rules, the

Packet-Buffer-Pool and the disk driver (which is considered to be an extension of the buffer pool space) must be trusted software.

From Section IV.C.2 the communication line drivers (see b. in section V.B.1) must be trusted tasks as they are responsible for calling the Audit and Marking packages. Manager.Q_Manager and Manager.Command_Processor (see e. above) also pass and receive critical information from the kernel and therefore must be trusted tasks.

As the transport tasks (c. above) do not have access to the internals of the buffers (which is hardware controlled) and any lost buffers will be eventually detected by the Auditor, they need not be trusted.

The last software group, Divider.Q (d. above) is more difficult to categorize. It does require read access to the header information of buffers going to the terminals, but since the Divider.Q cannot write or modify the buffers it can be considered to be just a more sophisticated transport task. Therefore, the software need not be trusted although it should be checked for covert channels because of its ability to read the header information.

V.C DESIGN OF THE TRUSTED AND UNTRUSTED SOFTWARE

V.C.1 TRUSTED SOFTWARE: STRUCTURE GRAPHS AND PSEUDO CODE

This section includes structure graphs and pseudo code for the software that was designated trusted in section V.B.2. The Packet-Buffer-Pool is depicted in figure V.C.1. Figure V.C.2 is the graph and pseudo code for one of the terminal drivers. The Console.Driver and Network-Interface.Driver are very similar to the terminal driver and are not coded. The active package Host is described in the two figures V.C.3.A. and V.C.3.B. The active package manager is described in three figures; the overall Manager in figure V.C.4.A., the Manager.Q-Manager in figure V.C.4.B., and the Manager.Command-Processor in Figure V.C.4.C.

Calls to the Packet_Buffer_Pool's Get entry are the only exceptions to the transport model discussed in sections II.D.5 and III.A.2 as Get may block the calling task.. This causes the Get entry to be a possible security problem. However, the Get entry can be used in different ways depending on the requirements for the system. Get can be used for flow control by limiting the number of buffers. If there are ample buffers to handle the expected traffic load, Get can be used to raise an alarm when an unexpected load or backlog is encountered. The reason for the embedded task Pool is to always accept the calling task. This prevent the calling task from timing out so that any action by the Packet_Buffer_Pool, if there are no buffers, cannot be bypassed by the calling task.

```

Task body Packet-Buffer-Pool is
Begin
  Loop
    Select
      Accept Put (P:Pointer) DO
        Pool.Put (P) End DO;
    OR
      Accept Get (P:Pointer) DO
        Pool.Get (P) End DO;
    End Select;
  End Loop;
End Packet-Buffer-Pool;

```

Task body Pool is ...

```

Begin
  Loop
    Select
      Accept Put (P:Pointer) DO
        Clear-Buffer;
        Queue-Pointer End DO;
    OR
      When buffer-count >0=>
        Accept Get (P:Pointer) DO
          Get-a-Buffer End DO;
    End Select;
  End Loop;
End Pool;

```

Figure V.C.1 Packet Buffer Pool

Task body In-Driver is

```

Begin Loop
  When Receive-Ready >0 =>
    Accept In (P:Out Pointer) DO
      Receive (P);
    End DO;

```

End In-Driver;

Task body Out-Driver is

```

Begin Loop
  When Line-Open >0=>
    Accept Out (P:In Pointer) DO
      Send (P);
    End DO;

```

End Out-Driver;

Package Body Receive (P:Pointer, Out);

```

Packet-Buffer-Pool.Get (P);
Fill (P); .. call to interface hardware
Audit.In (P.header, Return); ... kernel call
Check-Audit;
Marking.Check (P.header, Return); .. kernel call
Check-Marking;
End Receive;

```

```

Packet Body Send (P:Pointer, In);
Audit.Out (P.header, Return); .. kernel call
Marking.Check (P.header, Return); ... kernel call
Pass P to hardware
Buffer-Packet-Pool.Put (P);
End Send;

```

Figure V.C.2 Terminal Driver

Task body In-Driver is

```

Begin Loop
  When Host-Sends >0=>
    Accept In (Full:Out, Empty:In Host-Buffer) DO
      Full:=New-Host-Buffer;
      New-Host-Buffer:=Empty;
    End DO;

```

End In-Driver

Task body Out-Driver is

```

Begin Loop
  When Host-Available >0=>
    Compress.Full (Full, Empty);
    New-Host-Buffer:=Full;
    Clear (Full);
    Empty:=Full;
  End DO;

```

End Out-Driver;

Figure V.C.3.A Host.Driver

Task body Split is

```

Begin Loop
  In-Driver (New-Buffer, Return-Buffer); --be prepared to wait
  Split up host-buffer
  For I in 1 ... Number-of-Packets
    Begin
      Packet-Buffer-Pool.Get(P);
      Fill(P);
      Audit.In (P.header, Return);
      Check-Audit;
      Marking.Check (P.header, Return);
      Check-Marking;
    End For;
  End Split;
Task body Compress is, ...

```

```

Begin Loop
  Select
    When Host-Buffer-Not-Full >0=>
      Accept New (P: In Pointer) DO
        Audit.Out (P.Header, Return);
        Marking.Check (P.Header, Return);
        Put P in Buffer to Host;
        Buffer-Packer-Pool.Put(P); End DO;
    OR Accept Full (Full:Out, Empty:In Host-Buffer) DO
      Full:=Buffer-to-Host; Buffer-to-Host:=Empty;
    End Compress;

```

Figure V.C.3.B Host.Split and Host.Compress

(For structure graph see figure III.B.9)

Package Manager is

--active package

```
Procedure Send (P: In Pointer);
Procedure Terminal (P: Out Pointer);
Procedure Console (P: Out Pointer);
Procedure Network (P: Out Pointer);
Procedure Host (P: Out Pointer);
```

End Manager;

Package body Manager is

. . .

Task Q-Manager is

(see figure V.C.4.B)

. . .

Task Command-Processor is

(see figure V.C.4.C)

. . .

Procedure Send (P: In Pointer) is

Begin

 If P.Header.To = Front-End

 Then Command-Processor.Action (P);

 Else Q-Manager.In (P);

 End If;

End Receive;

Procedure Terminal (P: Out Pointer) is

Begin

 Q-Manager.Terminal (P);

 -- possible wait at entry

End Terminal;

. . .

(Procedures Console, Network, and Host are similar to Terminal)

End Manager;

Figure V.C.4.A Manager

Task body Q-Manager is

Begin Loop

Select

Accept In (P: In Pointer) DO

Queue in Q for P.Header.Receive or error

End DO;

OR

When Terminal-Q > 0 =>

Accept Terminal (P: Out Pointer) DO

De Queue (P);

Identification.Id (P.Header, Return); .. call to kernel

Check-Id;

Policy.Security-Check (P.Header, Return); .. call to kernel

Check-Policy;

End DO;

OR

When Console-Q > 0 =>

Accept Console (P: Out Pointer) DO

...

OR

When Network-Q > 0 =>

...

OR

When Host-Q > 0 => ...

End Q-Manager;

Figure V.C.4.B Manager.Q-Manager

```

Task body Command-Processor is
Begin Loop
  Accept Action (P: In Pointer) DO
    Audit.Out (P.Header, Return);
    Check-Audit;
    Extract Command;
    Case Command is
      when statistics => Begin
        Audit.Statistics (Request, Databack);
        .. reformat statistics into packets
        For I in 1 .. Number-of-Packets
          Begin
            Packet-Buffer-Pool.Get (P);
            .. enter statistics into P
            .. set up packet header
            Policy.Security-Check (P.Header, Return);
            Check-Policy;
            Audit-In (P.Header, Return);
            Check-Audit;
            Q-Manager.In (P);
          End; ... for
        End; .. Statistics
      when New-Password =>
        update Password file
    End Command-Processor
  End Loop

```

Figure V.C.4.C Manager.Command-Processor

V.C.2 NOT TRUSTED SOFTWARE: STRUCTURE GRAPHS AND PSEUDO CODE

The bulk of the software modules that are not trusted are simple transport tasks. They repeatedly wait at one entry for a packet and then pass the packet to a second entry. Their generic structure graph and pseudo codes are represented in Figure V.C.5.

The active package Divider is also a software module that need not be trusted. See Figure V.C.6.

Task Body Transport is
 Begin
 Loop
 First-Entry (...);
 Second-Entry (...);
 End Loop;
 End Transport;

Figure V.C.5 Generic Graph and Code For Transport Task

22
OF / DE

```

Task body Q is
Begin
  Loop
    Select
      Accept Send (P: Pointer) DO
        Queue on P.Header.Send.Address
      End DO:
    OR
      When Q1-Not_Empty > 0 =>
        Accept T1 ( P: Pointer ) DO
          Dequeue from Q1
        End DO;
      ( same for T2, T3, ... )
  End Loop
End Task

```

Figure V.C.6 Divider

VI EVALUATION OF THE SYSTEM DESIGN

VI.A BACKGROUND ON EVALUATION

VI.A.1 GENERAL

Generally in the past, system evaluations relied on informal methods such as testing, walk throughs, and modeling. In recent years more formalized methods have been developed based on mathematical principles. The formal evaluations can be divided into two methods, design verification and code verification.

Design verification shows the consistency between the model and the specifications. The assumptions are that the model is appropriate for the system and that the specification is complete. Some examples of these methodologies are FDM and HDM (see section II.C.3).

Code verification is the next step. It shows the consistency between the specification and the implementation. Again the assumptions are that the specification is appropriate and that the implementation language is correctly defined and implemented. Both EUCLID and GYPSY (see section II.C) are examples of programming languages that were written to assist with code verification.

A fair amount of controversy has arisen over the usefulness of formal verification. An article in Communications of the ACM [DEM79] argued that formal verifications would not play the same role in computer science as proofs do in mathematics. The article had the following to say about verification:

"What does the programmer know? He knows that his

VI.B THE TRUSTED COMPUTER SYSTEMS EVALUATION CRITERIA

The latest edition of the U.S. Department of Defense's Trusted Computer Systems Evaluation Criteria [CSC83] was published in August 1983 and has evolved from earlier U.S. evaluation studies. The criteria apply to both general purpose and embedded (or dedicated) computer systems. The purposes of the criteria are threefold (as defined in [CSC83]):

- * To provide users with a metric with which to evaluate the degree of trust that can be placed in computer systems for the secure processing of classified and other sensitive information.
- * To provide guidance to manufacturers as to what to build into their new and planned, trusted commercial products in order to provide widely available systems that satisfy trust requirements of the issuing agencies and/or for other sensitive applications.
- * To provide a basis for specifying security requirements in acquisition specifications."

The basic computer security requirement is to control access to information so that only authorized subjects can access it. From this basic requirement six fundamental requirements are derived. They are:

1. Security Policy
 - There must be an explicit and well-defined security policy enforced by the system.
2. Marking
 - Access control labels must be associated with

VI.B THE TRUSTED COMPUTER SYSTEMS EVALUATION CRITERIA

The latest edition of the U.S. Department of Defense's Trusted Computer Systems Evaluation Criteria [CSC83] was published in August 1983 and has evolved from earlier U.S. evaluation studies. The criteria apply to both general purpose and embedded (or dedicated) computer systems. The purposes of the criteria are threefold (as defined in [CSC83]):

- * To provide users with a metric with which to evaluate the degree of trust that can be placed in computer systems for the secure processing of classified and other sensitive information.
- * To provide guidance to manufacturers as to what to build into their new and planned, trusted commercial products in order to provide widely available systems that satisfy trust requirements of the issuing agencies and/or for other sensitive applications.
- * To provide a basis for specifying security requirements in acquisition specifications."

The basic computer security requirement is to control access to information so that only authorized subjects can access it. From this basic requirement six fundamental requirements are derived. They are:

1. Security Policy
 - There must be an explicit and well-defined security policy enforced by the system.
2. Marking
 - Access control labels must be associated with

objects.

3. Identification

- Individual subjects must be identified.

4. Accountability

- Audit information must be selectively kept and protected.

5. Assurance

- The secure computer system must contain hardware/software mechanisms that can be independently evaluated to provide sufficient assurance that they enforce the above four requirements.

6. Continuous Protection

- The trusted mechanism must be continuously protected against tampering and/or unauthorized changes.

The criteria is divided into four major divisions: A, B, C, and D ordered in a hierarchical structure with the highest division being A. The three higher divisions are subdivided into classes that are also hierarchical and are represented by a letter number pair. The ordering is:

beyond A1 > A1 > B3 > B2 > B1 > C2 > C1 > D.

Figure VI.B.1 shows the classes and the major improvements of each class over the class below it.

Also covered in the criteria is a guideline on covert channels. This attempts to provide system developers with an idea of just how high a "high" covert channel bandwidth is. One hundred bits per second is considered "high". One bit per second

is felt to be an acceptable rate in most application environments and there should be a capability to audit channels with bandwidths greater than one bit in ten seconds.

<u>Class</u>	<u>Major Improvement</u>	<u>Explanation</u>
Beyond A1	verified implementation	<ul style="list-style-type: none"> - formal verification at the source level, - covert timing channels are more fully addressed.
A1	verified design	<ul style="list-style-type: none"> - formal specification and design verification.
B3	security domains	<ul style="list-style-type: none"> - satisfy reference monitor requirements.
B2	structured protection	<ul style="list-style-type: none"> - formal security policy model, - covert channels are addressed.
B1	labeled security protection	<ul style="list-style-type: none"> - mandatory access control checks, - sensitivity labels are internally associated with subjects and objects.
C2	controlled access	<ul style="list-style-type: none"> - finely grained discretionary access control, - individually accountable users, - auditing of security related events.
C1	discretionary security protection	<ul style="list-style-type: none"> - separation of subjects and objects, - enforcing access limitations.
D	minimal protection	<ul style="list-style-type: none"> - systems that have been evaluated but fail to meet the requirements of a higher class.

Figure VI.B.1 Divisions and Classes

VI.C APPLICATION OF THE CRITERIA TO THE DESIGN IN
CHAPTERS III - IV

VI.C.1 OVERVIEW

Of the different possible approaches to applying the criteria, the one that seems to make the most sense is the bottom up approach. As all systems initially fit into the bottom division D, this is a good starting point. By meeting the requirements of a higher class the system can be moved up to that class. This continues until the system fails to meet the requirements of a class.

Another approach is to assume a division A system and work down until all the requirements are met. This is contrary to the philosophy of security which is that one assumes the least amount of information possible about a system.

An initial evaluation using the criteria should place the system in or close to its final class. After this the shortcomings and refinements can be addressed.

VI.C.2 IN DEPTH

The bottom up procedure will be used to attempt to find a possible upper rating of the front end security system. The requirements needed for the next level of design will be identified and expanded.

Division C provides for discretionary (need-to-know) protection and some auditing functions. Within this division there are two classes, C1 and C2.

A C1 system requires the following:

- a. define and control the access between users and objects;
- b. user identification and the protection of the authentication information;
- c. a domain for protection against tampering;
- d. periodic validation of the correct operation of the system;
- e. a security mechanism that is tested and working correctly;
- f. user documentation of the protection mechanism;
- g. other required documentation.

The security requirements in chapter II define "a" while the Audit package designed in chapter IV satisfies "b". Requirement "c" is covered by the hardware requirements in section II.E. Requirement "d" can be partially satisfied by hardware and also one of the functions of the package Audit is to check system integrity. "e", "f", and "g" are not requirements of the design phase; however, documentation must be started at this point. Considering the above, the front end system should fit within the C1 class.

A class C2 system needs the following additional requirements:

- a. the discretionary access control mechanism must provide per-user and system-wide defaults with granularity of a single user.
- b. storage objects must be written before they can be read.

- c. the trusted computing base (TCB) must be able to enforce individual accountability and audit individuals.
- d. the TCB must be able to create, maintain, and protect a record of accesses to the objects it protects.
- e. the TCB must encapsulate the resources to be protected.
- f. testing must include a search for obvious flaws that would allow violation of resource encapsulation or unauthorized access to the audit or authentication data.
- g. additional documentation.

The objects protected by the front end are packets that are switched by the front end. Requirement "a" is only a concern as far as who has access to the audit and security information. This is well defined in chapter II. The packet buffer pool manager is responsible for enforcing "b". The driver tasks and Audit package enforce requirements "c" and "d". As access to packets is strictly controlled by the packet buffer pool manager "e" is also satisfied. As in a class C1 system, "f" and "g" are not strict requirements of the design phase. Therefore, the system should be able to meet the C2 requirements.

Division B provides for mandatory protection by using sensitivity labels and the reference monitor concept. There are three classes, B1, B2, and B3, within this division.

A B1 system requires the following additions to a C2

system:

- a. sensitivity labels for all subjects and objects.
- b. the enforcement of a mandatory access control policy.
- c. auditing based on object security levels.
- d. the security mechanism must be independently tested and found to work as claimed.
- e. an informal or formal model of the security policy that can be defended as being consistent with its axioms.
- f. additional documentation.

The requirements "a" and "b" are defined by the security model (section II.E) and are enforced by the kernel entries:

Identification and Policy. The package Audit will enforce requirement "c". The testing of the security mechanism, "d", is not performed during the design phase. Requirement "e" is not well defined. An informal model is presented in section II.E along with a justification based on current security principles. As in the C2 class, requirement "f" is met during a subsequent phase. Therefore, only item "e" could cause problems for the system to be in the B1 class. Since "e" needs to only be met informally this should not be a major hurdle.

A class B2 system requires the following additions to a B1 system:

- a. storage objects must be purged of data before reuse.
- b. sensitivity labels are extended to include all EDP system resources.

- c. the TCB must notify a user of each change in the working security level associated with that user during an interactive session and a user can query the TCB for the current security level.
- d. the mandatory access control policy is extended to include all resources by subjects.
- e. the TCB must support a trusted communication path between itself and the user.
- f. the TCB must be able to audit identified covert storage channels.
- g. new system architecture features must be included to protect against tampering.
- h. a thorough search must be conducted for covert channels.
- i. a trusted facility management must be supported.
- j. the TCB must be found relatively resistant to penetration and all discovered flaws must be corrected.
- k. testing must demonstrate that the TCB implementation is consistent with top-level specifications.
- l. a formal model of the security policy must be maintained and proven consistent with its axioms.
- m. a descriptive top-level specification must be maintained and shown to be a true description of the TCB interface.
- n. a configuration management system must be used.
- o. additional documentation.

The design of a B2 system is much more tightly controlled than a B1 system. Requirements "f", "h", "i", "j", "l", and "n" have not been addressed in this design and would require a great deal of extra work and a number of additional support tools. These would include a configuration management system, a methodology for proving consistency between a formal model and its axioms, a penetration analysis tool, and a covert channel analysis tool.

A class B3 system must satisfy the reference monitor requirements. A security administrator is also needed, audit mechanisms must be expanded, and system recovery procedures are required. A B3 system must be highly resistant to penetration.

A class A1 system is functionally equivalent to a B3 system. The distinguishing feature is that formal specification and design verification techniques must be used. This will result in a higher degree of assurance.

Systems beyond class A1 are not fully specified as yet. They are envisioned to include formal verification at the source code level. Consideration must also be given to the correctness of the tools used for design and implementation (compilers, assemblers, loaders, etc.).

VII CONCLUSIONS AND FUTURE WORK

VII.A CONCLUSIONS

Each of the three objectives defined in section I.B.4 was met to some degree in this thesis.

Objective a., a case study on the design of a secure front end system, is presented in chapters I thru VI. The different functions performed by the front end separate easily into tasks and packages. The modularity of these tasks and packages permit simple and well-defined interfaces. Although concurrency, which is a functional necessity, is considered to be a very difficult construct to validate or verify, it does add greatly to the conceptual image and simplicity of the design. For example, the diagrams are easy to comprehend and one can get an overall picture of the system quickly.

A variety of tradeoffs for security in both hardware and software have arisen. One of the more obvious tradeoffs for security is speed. Most of the routines in the security kernel perform only checking functions, but these still require computer time. This is becoming less important as hardware becomes cheaper, faster, and smaller and more of the security-related functions are directly supported by hardware. This has already occurred in processors such as the Honeywell SCOMP [CS182], which includes special hardware called the security protection module (SPM). The SPM resides between the processor and all other system elements. This enables the SPM to capture all processor requests and perform the required security functions in the hardware. Another tradeoff is simplicity over efficiency. Because

simplicity is required for verification this tradeoff is inevitable. Again, however, this becomes less significant as hardware costs decrease. Systems need to be simplified so that their trustworthiness, reliability, dependability, and limits of operation can be checked intuitively.

Objective b., the case study of the application of the design methodology, worked well in constructing a reference monitor based system. The security kernel can be represented as a single package that is called as required. Although ADA itself is a large and sometimes cumbersome language, the subset used in this design was compact and simple.

The last objective, c., was to understand how to apply the evaluation criteria to the front end system. The evaluation criteria were applied very early in the design. They were not easy to apply as they assume a completed system and they leave a great deal to interpretation. The boundaries between classes are well defined but some of the conditions within the classes are not, and may be interpreted broadly. For example, the security testing in a B2 system (section 3.2.3.2.1) states "The TCB must be found relatively resistant to penetration. All discovered flaws must be corrected." The word "relatively" can take on a wide range of meanings. A more quantitative measure is needed for clarity. Another example is in section 3.1.3.2.2 where the words "shown to be consistent" are used.

The Trusted Computer System Evaluation Criteria is written for "ADP systems being considered for the processing and/or storage and retrieval of sensitive or classified information"

[CSC83]. There seems to be little consideration given to real-time systems and to issues such as efficiency and reliability. Efficiency and reliability are not normally considered to be security issues. In real-time systems they come under the security objective 'availability of service' (see section II.B.2). Therefore, they both should be considered in the design, implementation, and evaluation of a real-time system such as the front end.

The evaluation in chapter VI indicates a level of trust that should be feasible to reach. Therefore, the secure front end system should be designed to meet the B1 system criteria. The effort required to upgrade to a B2 or higher system would be substantial.

VII.B DIRECTION OF FUTURE WORK

Future study is needed in many of the areas discussed in this thesis. These include the design methodology; embedding Euclid in ADA; computer-aided design with ADA and Euclid; mapping between design and implementation, and implementation issues.

Work needs to be done on defining a minimum usable subset of ADA that can be utilized in the design methodology. This subset should be formally specified so that verification techniques can be applied to it.

Research should be continued on embedding Euclid in ADA. This will expand the range of applications in which Euclid can be used.

A new area of work to parallel that on computer-aided design with ADA is computer-aided design with Euclid. This should simplify designing in Euclid.

The topic of the mapping between design and implementation has always been a difficult one. Advances in computer tools for software generation of code are needed for the following:

- a. comparing of human generated code and the graphic design.
- b. Comparing the data flow in the graphic design method and the data flow in the implemented code.
- c. defining the mapping and the partitioning of the design onto hardware and software.

BRIEF DESCRIPTION OF THE DESIGN METHOD

The design method that is used is described in [BUH84]. Systems are described in terms of black boxes using blueprint-like pictures. The black boxes represent ADA packages, tasks, and procedures. Since the internal software of a black box is not visible, any language can be used internally as long as the interfaces are kept consistent.

The basic components of the blueprints are:

Packages

(rectangles)

Tasks

(parallelograms)

Unspecified
module

(cloud)

Data

Access connection

Data flow

EXAMPLES OF CODING

Fixed Order
Accepts

```
accept A do ... end;
```

```
·  
·  
·  
accept B do ... end;
```

Selective Wait

```
select  
accept A do ... end;
```

```
·  
·  
or  
accept B do ... end;  
end select;
```

Conditional Wait

```
when x = >  
accept A do ... end;
```

BIBLIOGRAPHY

- [ART81]** Advances in Computer System Security.
Edited by Rein Turn. Artech House Inc.,
Dedham, MA. 1981.
- [BAR80]** Barnes, J.G.P. An Overview of Ada.
Software Practice and Experience.
Vol. 10. 1980. p.p. 851-887
- [BOY81]** The correctness problem in computer science.
Edited by R.S. Boyer and J.S. Moore.
Academic Press Inc., London, England, 1981.
- [BUH84]** Buhr, R.J.A. System design With ADA.
Prentice Hall, Englewood Cliffs, New Jersey,
1984
- [CAR84]** Carter, T.M., et al. Transforming an ADA
Program Unit to Silicon and Verifying its
Behavior in an ADA Environment: A First
Experiment. University of Utah, paper to be
presented at Compcon Spring Conf. San Francisco,
CA. '84, Feb 28-Mar 1, 1984.
- [CCI80]** Interface Between Data Terminal Equipment
(DTE) and Data Circuit Terminating Equipment
(DCE) for Terminals Operating In the Packet
Mode On Public Data Networks. CCITT. Geneva,
Switzerland. 1980
- [CHE81]** Cheheyli, M.H., et al. Verifying Security.
ACM Computing Surveys, Vol. 13, No. 3.
1981. pp. 279-339.
- [COR81]** Cordy, J.R. and Holt, R.C. Specification of
concurrent Euclid. Technical Report
CSRG-133, Version 1, CSRG University of
Toronto, Toronto, Ont. August 1981
- [CRO82]** Crowe, D.R. Ottawa Euclid Reference Manual.
Technical Report TR-5613-81-7 I.P. Sharp
Assoc. Ltd., Ottawa, Ont. Dec. 1982
- [CSC83]** Trusted computer system evaluation criteria.
DOD Computer Security Center, August 1983.
- [CSI80]** The Department of Defence Computer Security
Initiative, 2nd seminar. Computer Security
Initiative, National Bureau of Standards,
Maryland. January 1980.

- [CSI82]** Proceedings of the Fifth Seminar On the DOD Computer Security Initiative. National Bureau of Standards, Gaithersburg, MA. May 1982.
- [DAH72]** Dahl, O.J., Dijkstra E.W., and Hoare C.A.R. Structured Programming. Academic Press Inc., New York, N.Y. 1972.
- [DEM79]** DeMillo, R.A. et al. Social Processes and Proofs of Theorems and Programs. Communications of the ACM. Vol. 22, No. 5, 1979. pp. 271-280.
- [DEN79]** Denning, D.E. and Denning P.J. Data Security. ACM Computing Surveys, Vol. 11, No. 3, 1979. pp. 227-249.
- [DOD82]** Reference Manual For the ADA Programming Language. Revised draft MIL-STD 1815. U.S. Department of Defense. July 1982.
- [FIP77]** Data Encryption Standard, Federal Information Processing Standards (FIPS) Publ. 46. National Bureau of Standards. 1977.
- [FIP80]** Guidelines For Security of Computer Applications, FIPS PUB 73. National Bureau of Standards. June 1980.
- [FIP82]** Standard On Computer Data Integrity, FIPS Prepublication. National Bureau of Standards. 1982.
- [HOA74]** Hoare, C.A.R. Monitors: An Operating System Structuring Concept. Communications of the ACM Vol. 17, No. 10, 1974. pp. 549-557.
- [HOL81]** Holt, R.C. A Short Introduction to Concurrent Euclid. Technical Note #26, CSRG. University of Toronto, Toronto, Ont. September 1981.
- [INT82]** ADA and Euclid as Programming Languages for Communication System - A Feasibility Study. Intellitech Canada Ltd., Ottawa, Ont. 1982.
- [INT83]** INTEL IAPX 286 programmer's Reference Manual order number 210498-001. Intel Corporation, Santa Clara, California, 1983.

- [IPS80]** Toronto Euclid Language Specifications. DOC TR-3819-80-2. I.P. Sharp Associates Limited, Ottawa, Ont. January 1980.
- [IPS82]** A Blueprint for a Verification and Evaluation Environment based upon EUCLID. FR-5017-1982-01. David Bonyun et al, I.P. Sharp Assoc. Ltd., Ottawa, Canada, March 1982.
- [JEN75]** Jensen, K. and Wirth, N. Pascal User Manual and Report. 2nd ed. Springer-Verlag, New York, NY. 1975.
- [LAM77]** Lampson, B.W. et al. Report On The Programming Language Euclid. SIGPLAN Notices. February 1977.
- [LAN81]** Landwehr, C.E. Formal Models for Computer Security. ACM Computing Surveys, Vol. 13, No. 3, 1981. pp. 247-278.
- [LON78]** London, R.L., et al. Proof Rules for the Programming Language Euclid. Acta Information, 10, pp. 1-26, 1978.
- [MAH81]** Mahjoub, A. Some Comments On Ada as a real-time programming language. SIGPLAN Notices. Vol. 16, No. 2, 1981. pp. 89-95.
- [MYE80]** Myers, P.A. Subversion: The neglected aspect of computer security. Ph.D. Thesis. Naval Postgraduate School, Monterey, CA. June 1980.
- [NBS80]** The SRI Hierarchical Development Methodology (HDM) and its Application to the Development of Secure Software. NBS Special Publication No. 500-67. U.S. Department of Commerce, Washington, DC. October 1980.
- [NBS81]** Validation, Verification, and Testing of Computer Software. NBS Special Publication No. 500-75. U.S. Department of Commerce. February 1981.
- [NOR82]** Computers and Security. Vol. 1, No. 1. North-Holland Publishing Company, Amsterdam, The Netherlands. January 1982.

- [NYE81]** Nye, J.M. Who, what, and where in communications security. Marketing Consultants International Inc., 100 West Washington St., Hagerstown, MA. April 1981.
- [PYL81]** Pyle, I.C. The Ada programming language. Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
- [SCH79]** Schell, R.R. Computer security, the Achilles' heel of the electronic Air Force? Air University Review, Vol. 30, no. 2, 1979.
- [SHA77]** Shankar, K.S. The total computer security problem: An overview. Computer, June 1977, pp. 50-73.
- [SIL81]** Silberschatzy, A. On the synchronization mechanism of the Ada language. SIGPLAN Notices. Vol. 16, No. 2, 1981. pp. 96-103.
- [SNY81]** Snyder, L. Formal Models of capability-based protection systems. IEEE Transactions On Computers. Vol. C-30, No. 3, 1981.
- [STR80]** Stroet, J. An alternative to the communication primitives in Ada. SIGPLAN Notices. Vol. 15, No. 12, 1980.
- [STU82]** Stubblebine, A.N. Electronic warfare: A key ingredient for "Speechlessness". Signal Magazine. April 1982. pp. 15-17.
- [TUR81]** Turn, R. Trusted computer systems needs and incentives for use in government and the private sector. Report No. R-2811-DR&E, Rand Corporation, Santa Monica, CA. June 1981.
- [VAN80]** Van de Bos, J. Comments on ADA process communication. SIGPLAN Notices. Vol. 15, No. 6, June 1980.
- [WOR79]** Wortman, D.B. On legality assertions in Euclid. IEEE Transactions on Software Engineering. Vol. SE-5, No. 4, 1979. pp. 359-367.
- [YOU79]** Yourdon, E. and Constantine, L.L. Structured design fundamentals of a discipline of computer program and systems design. Prentice-Hall Inc., 1979.
- [YOU81]** Young, W.D., and Good, D.I. Steelman and the verifiability of (preliminary) ADA. SIGPLAN Notices. Vol. 15, No. 2, 1981.

END

11-06-86

FIN