NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN OF A TRUSTED COMPUTING BASE EXTENSION FOR COMMERCIAL OFF-THE-SHELF WORKSTATIONS (TCBE)

by

Jason X. Hackerson

September 1998

Thesis Advisor: Cynthia Irvine

Approved for public release; distribution is unlimited

19981103 059

DTIC QUALITY INSPECTED 4

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1998.	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE DESIGN OF A TRUSTED COMPUTING BASE EXTENSION FOR COMMERCIAL OFF-THE-SHELF WORKSTATIONS (TCBE)	5. FUNDING NUMBERS
6. AUTHOR(S) Jason X. Hackerson	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

United States policy requires that access to and dissemination of classified information be controlled. Separate networks and workstations for each classification do not meet user requirements. Users also need commercially available office productivity tools. Traditional multilevel systems are costly and are unable support an evolving suite of Commercial Off-The-Shelf (COTS) applications.

This thesis presents a design for a Trusted Computing Base Extension (TCBE) that allows COTS workstations to function securely as part of a multilevel network that uses high assurance multilevel servers as the backbone. The TCBE will allow COTS workstations to use commercially available software applications, while providing a Trusted Path to a high assurance multilevel server.

The research resulted in a design of a TCBE system that can be employed with COTS workstations, allowing them to function as untrusted clients in the context of a secure multilevel network.

14. SUBJECT TERMS Information Assurance, Multilevel Security, MLS, Secure LAN, Trusted Computing Base, Trusted Path	15. NUMBER OF PAGES 168
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFI-CATION OF THIS PAGE	19. SECURITY CLASSIFICA-TION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

Approved for public release; distribution is unlimited

DESIGN OF A TRUSTED COMPUTING BASE EXTENSION FOR COMMERCIAL OFF-THE-SHELF WORKSTATIONS(TCBE)

Jason X. Hackerson
Captain, United States Marine Corps
B.S., United States Naval Academy, 1991

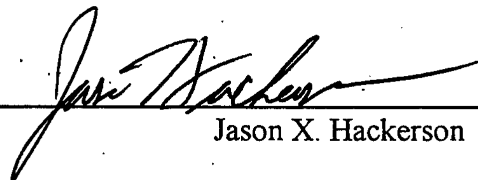
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1998

Author:

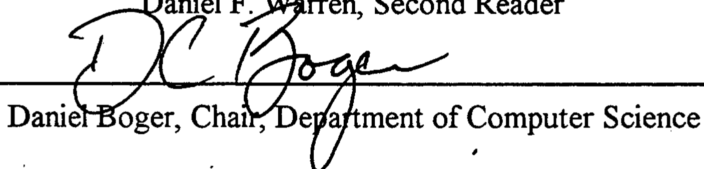


Jason X. Hackerson

Approved by:

Cynthia E. Irvine, Thesis Advisor

Daniel F. Warren, Second Reader



Daniel Boger, Chair, Department of Computer Science

ABSTRACT

United States policy requires that access to and dissemination of classified information be controlled. Separate networks and workstations for each classification do not meet user requirements. Users also need commercially available office productivity tools. Traditional multilevel systems are costly and are unable support an evolving suite of Commercial Off-The-Shelf (COTS) applications.

This thesis presents a design for a Trusted Computing Base Extension (TCBE) that allows COTS workstations to function securely as part of a multilevel network that uses high assurance multilevel servers as the backbone. The TCBE will allow COTS workstations to use commercially available software applications, while providing a Trusted Path to a high assurance multilevel server.

The research resulted in a design of a TCBE system that can be employed with COTS workstations, allowing them to function as untrusted clients in the context of a secure multilevel network.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. PURPOSE 1
B. SCOPE 3
C. OVERVIEW OF CHAPTERS 3
I. Introduction 3
II. TCBE Functional Requirements for a Secure Multilevel Network 4
III. TCBE System Specification Development 4
IV. Software Implementation of TCBE components 4
V. Hardware Implementation of TCBE components 4
VI. Conclusions 5
D. OVERVIEW OF APPENDICES 5
1. Appendix A - TCBE Design Development 5
2. Appendix B – TCBE Software Specification Document 5
3. Appendix C – TCBE Executive Source Code 5

II. TCBE FUNCTIONAL REQUIREMENTS FOR A SECURE MULTILEVEL NETWORK 7

A. TCBE FUNCTIONAL REQUIREMENT 1 7
B. TCBE FUNCTIONAL REQUIREMENT 2 8
C. TCBE FUNCTIONAL REQUIREMENT 3 9
D. TCBE FUNCTIONAL REQUIREMENT 4 9
E. TCBE FUNCTIONAL REQUIREMENT 5 10

III. TCBE SYSTEM SPECIFICATION DEVELOPMENT 11

A. CONCEPT OF OPERATIONS 12
1. Initialization 12
2. Trusted Operations 12
3. Data Path Operation 13
4. Trusted Path Options 13
5. TCBE Physical Structure 14
B. OUTLINE OF THE BASIC MODULES 15
1. TCBE Core 16
2. Trusted Path Module 16
3. Data Path Module 17
4. Keyboard Secure Attention Key Module 18
5. Encryption Module 18
6. TCBE Hardware 19
C. LAYER DESCRIPTIONS 19

1. Hardware Layer 20
2. Hardware Management layer 20
3. Encryption Layer 20
4. SAK Layer 21
5. Trusted Path Layer 21
6. Data Path Layer 21
7. TCBE Main Layer 21

IV. SOFTWARE IMPLEMENTATION OF TCBE COMPONENTS 23

A. TCBE SOFTWARE PLACEMENT CONSIDERATIONS 23
1. TCBE Software Implementation Option 1 24
2. TCBE Software Implementation Option 2 25
3. TCBE Software Implementation Option 3 27
B. TCBE OPERATING SYSTEM DESIGN 28
1. TCBE Executive Requirements 28
2. TCBE State Operations 29

V. HARDWARE IMPLEMENTATION OF TCBE COMPONENTS 39

A. TCBE HARDWARE REQUIREMENTS 39
1. Memory 39
2. Microprocessor 41
3. Communications Interfaces 41
B. HOST COMPUTER HARDWARE REQUIREMENTS 42
1. Hard Disk Drive Presence 42
2. Amount of Random Access Memory 45
C. IMPLEMENTATION OF TCBE 45

VI. CONCLUSIONS 49

A. COST EFFECTIVENESS 49
B. RECOMMENDATIONS FOR FUTURE RESEARCH 49
1. Host Computer 49
2. Trusted Path Research 51
C. CONCLUSIONS 52

APPENDIX A. TCBE DESIGN DOCUMENTATION 55

A. TCBE SCENARIOS 56
B. TCBE STATE DIAGRAMS 61
C. TCBE OBJECT DIAGRAM 63
D. TCBE FUNCTIONAL MODULES DIAGRAM AND DEFINITIONS 64

APPENDIX B. TCBE SOFTWARE SPECIFICATION DOCUMENT 69

1. TRUSTED COMPUTING BASE EXTENSION SOFTWARE DESCRIPTION 69
2. TRUSTED COMPUTING BASE EXTENSION LAYERS 69
3. TRUSTED COMPUTING BASE EXTENSION MODULES 70

APPENDIX C. TRUSTED COMPUTING BASE EXTENSION EXECUTIVE SOURCE CODE 87

1. IMPLEMENTATION FUTURE WORK 87
a. Interrupt Service Routines 87
b. Memory Manager 88
2. ASSEMBLY LANGUAGE TO C LANGUAGE LINKAGE 88
a. Resolving Memory and Linkage Issues 88
b. Resolving Shared Procedure and Function Names 89
3. TCBE IMPLEMENTATION SOURCE CODE 91
a. Custom.h 91
b. TCBEX.h 92
c. TCBEX.c 97
d. TCBE_Module.h 112
e. TCBE_Module.c 113
f. TP_Module.h 115
g. TP_Module.c 115
h. Stddefs.asm 118
i. Regs.asm 129
j. Makefile 148
k. Batch File 150

LIST OF REFERENCES 151

BIBLIOGRAPHY 153

INITIAL DISTRIBUTION LIST 155

LIST OF FIGURES

Figure 1 TCBE Layer Diagram 20
Figure 2 TCBE Software Implementation Options 24
Figure 3 State Machine Engine 30
Figure 4 TCBE State Diagram 31
Figure 5 Trusted Path State Diagram 32
Figure 6 Data Path State Diagram 34
Figure 7 TCBE State Matrix 35

I. INTRODUCTION

A. PURPOSE

Access to multiple levels of information is essential to the operation of many Department of Defense (DoD) and Department of the Navy (DON) organizations. The ability to access the information via a single computer terminal is cost effective and efficient. Computer systems that allow the user to securely access different levels of classified material from a single terminal while maintaining the ability to use popular Commercial Off The Shelf (COTS) hardware and software do not exist.

Currently, the DoD allows access to multiple levels of information on automatic data processing systems by following a "system-high" policy of classifying computers and using "guard" computers to separate networks. A system-high policy states that all data that is processed on system high equipment is considered labeled at the classification of the computer. For example, if a computer is labeled as a system-high or top secret processor, then any individual who is authorized to use that computer must have a clearance level as high as the computer, even if that person uses the computer only to access and process unclassified material.

Secure Department of Defense networks are maintained for each classification level. Guard computers link high and low networks, such as the SIPRNET and the NIPRNET. These computers permit certain, controlled transfers of data between networks such as the movement of unclassified electronic mail to high networks. The guard ensures that the networks at each security level are correctly separated, but it does

not allow simultaneous multilevel access to the information. Most guards currently in use do not label data or prevent Trojan horses that may downgrade sensitive information without the user's knowledge. Additionally, multiple networks require that duplicate equipment to be installed and maintained within a single site. Many users of these systems must maintain a separate end terminal for each classification level on their desks in order to access the different classes of information.

There are high assurance computer systems that allow multiple levels of information to reside on the same machine. These systems are commercially available and are purchased by the DoD primarily to function as guard platforms. High assurance computers are costly to develop, purchase, and maintain. Their interfaces do not support commonly used commercial software (e.g. Microsoft products, or many popular UNIX applications). Additionally, since the cost of these systems is high, they cannot be placed on each desktop.

To eliminate redundancy and cost, a solution to these problems that uses a few current high assurance systems as servers and many COTS computers as end terminals to provide a single secure multilevel local area network (LAN) should be found. By providing COTS solutions for the client end of a secure multilevel LAN, users will be able to take advantage of the security features in a high assurance system to process sensitive and classified data while using popular commercial software.

The design of a Trusted Computing Base Extension (TCBE) for COTS clients is a key part of a secure multilevel network. The TCBE helps ensure that the information processed is not compromised, and that it comes from a secure resource. Additionally, the

trusted path enabled by the TCBE at the client provides the user the capability to efficiently process all levels of information.

B. SCOPE

The objectives of this thesis are:

- Development of Trusted Computing Base Extension (TCBE) functional requirements for COTS clients of a Secure Multilevel Network.
- System Specification documenting the design of a TCBE for COTS personal computers.
- Development of a prototype TCBE for a COTS personal computer.

C. OVERVIEW OF CHAPTERS

I. Introduction

Chapter I discusses the purpose and scope of the thesis. An overview of the following additional chapters is provided: Chapter II, TCBE Functional Requirements for a Secure Multilevel Network; Chapter III, TCBE System Specification Development; Chapter IV, Software Implementation of TCBE components; Chapter V, Hardware Implementation of TCBE components; Chapter VI, Conclusions and Recommendations; APPENDIX A. TCBE Design Development; APPENDIX B. TCBE Software Specification Document; APPENDIX C. TCBE Executive Source Code.

II. TCBE Functional Requirements for a Secure Multilevel Network

TCBE requirements will be developed following the Department of Defense's Trusted Computer Security Evaluation Criteria (TCSEC) for Class B3 systems [1]. The requirements that the TCBE must meet in order to operate as part of the multilevel network are addressed. The purpose of the TCBE is not to build a Class B3 system at the client, but to extend the evaluated Class B3 system that provides the locus of security control for the multilevel secure local area network to the client systems.

III. TCBE System Specification Development

Chapter III will document the development of the system specification and the design documentation for the TCBE. An analysis of the functional requirements will be discussed, a concept of operations will be developed, and the TCBE's basic operating modules will be outlined.

IV. Software Implementation of TCBE components

This chapter explains the design and implementation of each of the proposed TCBE's modules. This chapter also explains how COTS software will be integrated into the TCBE framework.

V. Hardware Implementation of TCBE components

This chapter describes the hardware that was used and modified to reach the TCBE's security policy requirements.

VI. Conclusions

Chapter VI contains the conclusions for the use and future development of the Trusted Computing Base Extension. The chapter focuses on the cost effectiveness of implementing the TCBE in a secure network environment and provides recommendations into future research areas involving the design and development of Trusted Computing Base Extensions.

D. OVERVIEW OF APPENDICES

1. Appendix A - TCBE Design Development

This appendix provides the documents that were developed in the course of designing the Trusted Computing Base Extension. The first document contains the scenarios of TCBE operation. The next set of documents contains the state diagrams that were developed from the scenarios. The system's object diagram was produced next along with the system's layers and function modules.

2. Appendix B – TCBE Software Specification Document

This appendix provides the TCBE Software Specification Document. This document provides descriptions of the major modules and functions of the Trusted Computing Base Extension.

3. Appendix C – TCBE Executive Source Code

This appendix provides the source code and all files to implement the TCBE subset.

II. TCBE FUNCTIONAL REQUIREMENTS FOR A SECURE MULTILEVEL NETWORK

The goal of the TCBE is to allow the user the ability to access multiple levels of information via a single terminal device. As such, it will have the characteristics of the reference monitor concept: the TCBE is always invoked, it is tamperproof, and that the Trusted Computing Base Extension code is analyzable for correctness and security.

The TCBE will be established through modules and hardware that are attached to the client operating system and computer without modification to the operating system source code. By avoiding source code modification of the operating system, the design of the TCBE is simplified and its security processing is more easily evaluated. The TCBE will be considered as a volatile device that functions as a robust client without permanent data storage capabilities. The TCBE has the following requirements:

A. TCBE FUNCTIONAL REQUIREMENT 1

The TCBE will prevent object reuse and data remanence. Object reuse is described as the ability of an unauthorized user to access data left from a previous user's session [2]. The data could remain in memory, accessed through shared memory, or be stored inadvertently on a secondary storage device such as a hard drive. To satisfy the object reuse requirement, the computer system must make provisions to ensure that all potentially shared objects such as data segments are clear of data or have been overwritten by a random pattern of bits.

"Data Remanence is the residual physical representation of data that has been in some way erased"[3]. Information still remains on storage media even after a user

application or operating system has called for the data to be deleted. Data can be cleared from storage devices by overwriting the information such that normal operation of the system does not allow the information to be accessed. Information can be purged from a system by using physical methods such as a degausser which magnetically moves the storage media from a "recorded state to an unrecorded state" [3]. Ineffectively cleared memory and storage devices can lead to unauthorized object reuse after a change in session level or between users.

Client workstations will be assumed to be under the physical control of a single user during a session. That user will logon and negotiate a session level. All data connected with that session will be purged from the client system by the TCBE at the end of the operation of the negotiated session level. User files will not be stored on the TCBE after the user has completed a session. Operating system and application software for the TCBE will reside on read-only media. Data the operating system uses for normal operations will reside in volatile storage devices whose memory can be erased or quickly overwritten at the end of a session.

B. TCBE FUNCTIONAL REQUIREMENT 2

The High Assurance Server will enforce the system's mandatory access control policy. The TCBE will only gain access to objects according to the policy that the High Assurance Server enforces. The High Assurance Server will mediate all user reads and writes from the TCBE.

C. TCBE FUNCTIONAL REQUIREMENT 3

The TCBE will support a Trusted Path for initial login and authentication. The Trusted Path provides assurance to the user that the TCBE is connected to the trusted server, and it provides the assurance to the trusted server that it is connected to an authorized client. Communications via this path shall be activated by the user or the TCB and shall be logically isolated and unmistakably distinguishable from other paths. The user will be able to invoke the Trusted Path via a "secure attention key" (SAK) at any time in order to communicate directly with the high assurance server. When the Trusted Path is invoked, all client processing, except the trusted path interface on the TCBE, will be halted. The user will be able to switch session levels or log out once the Trusted Path is invoked.

Identification and authentication of a single user will be initiated from the user's terminal via the TCBE. The authorization database will reside at the High Assurance Server. The TCBE will provide only user name and identification information such as passwords and biometric data to the server. After identification and authentication is complete, the user will have access to the objects (files) that the user is authorized for.

D. TCBE FUNCTIONAL REQUIREMENT 4

The TCBE will consist of functional modules and well-defined interfaces that minimize the complexity of the TCBE. The TCBE shall maintain a domain for its own execution that protects it from external interference or tampering. By placing the TCBE on a controller board that manages the client computer's input and output functions, logical and physical separation from the client computer's operating system applications is

achieved. The TCBE will be designed so that unauthorized personnel can not modify its software modules. The system administrator or security officer will physically control the configuration management of the client computer operating system and applications.

The TCBE will be designed so that it will support trusted recovery to a known secure state in the event of system failure. When loss of power occurs, all information that resided on the TCBE client shall be erased. When the TCBE is started again, it will initiate procedures that prevent object reuse and data remanence from the interrupted session.

E. TCBE FUNCTIONAL REQUIREMENT 5

Interfaces to the TCBE will be well defined and non-bypassable. All secure access to the high assurance server from the client computer will be via the TCBE. The protocol established between the TCBE and the high assurance server will ensure that clients without TCBE support will be unable to obtain service from the server.

III. TCBE SYSTEM SPECIFICATION DEVELOPMENT

System development of the TCBE will begin with a description of the requirements of the system, followed by analysis of the functional requirements that will produce a concept of operations for the TCBE as well as form the basis of the system's modules. Once the basic modules have been outlined, the system will be broken into layers. The layers are formed by presenting the system in a bottom up manner, where the elements of the system at the higher layer require the functions and information that are provided by the elements at the lower layers in order to conduct their operations. Layering indicates which modules of the system have the greatest access to the system's resources and thus are conceptually more privileged. Thus an architecture in which objects are allocated to hierarchically ordered layers is achieved.

By breaking the TCBE into layers, we ensure that data dependencies travel from the least privileged modules to the most privileged modules. Additionally, layering of the system allows the designer to refine the basic modules and break them down into layers of smaller modules that have well defined interfaces and are easier to implement. Since it is easier to demonstrate that smaller modules are performing in the manner desired, the designer's confidence that the TCBE will perform correctly is increased.

Following the layering, the modules and their databases are designed. Interfaces, input and output parameters and actual functions are defined. The designs of the software and hardware modules are discussed in Chapters IV and V.

A. CONCEPT OF OPERATIONS

A concept for how the TCBE will operate is developed by examining the functional requirements. An analysis of the requirements leads to operational scenarios and state diagrams for the TCBE. A brief description of the TCBE Concept of Operations is provided, the actual scenarios and state diagrams are presented in Appendix A.

1. Initialization

The TCBE will begin operations in an Initialization state. In this state the TCBE and its modules are loaded and the host computer's random access memory (RAM) is overwritten. At this point, the TCBE is in a wait state and all that is displayed to the user is a blank screen or a TCBE logo.

2. Trusted Operations

The trusted operation state begins whenever the user presses the Secure Attention Key (SAK). The Trusted Path activates and begins an initialization sequence with the high assurance server (HAS). The initialization sequence first establishes that the TCBE is talking to the HAS and the HAS server provides the TCBE with the encryption key that it will use to encrypt further communications with the HAS along the Trusted Path. This trusted path encryption key will encrypt all communications whenever the TCBE enters the trusted operation state.

Once the TCBE has received the trusted path encryption key, the user will be able to login to the HAS. The user logs in by entering in a User-ID and password, as well as any other authentication information such as a smart card or biometric data. The TCBE encrypts this data with the Trusted Path encryption key and sends it to the high assurance

server for processing. Once the HAS authenticates the user, it sends a session encryption key to the TCBE. The session encryption key will be used to encrypt the data transfers between the host computer and the HAS for the selected session level. When the login procedure is completed, the user will be able to begin work at the selected session level, switch session levels, return to the current session after pressing the SAK, or log out of the session.

3. Data Path Operation

When the user begins work at the selected session level, the TCBE moves into a data path state. In this mode the TCBE acts only as conduit for the secure movement of data between the HAS and the host computer. Before operations in this state begin the TCBE will ensure that the host computer's operating system is loaded and that the host computer CPU is ready to begin operations. Once those tasks are complete the TCBE is ready to encrypt and decrypt data and send it to or receive data from the high assurance server. Temporary reads and writes such as swap files are sent to the volatile secondary storage unit within the host computer.

4. Trusted Path Options

The user can press the Secure Attention Key at any point. If the SAK is pressed while the TCBE is in the data path state, the TCBE halts the host computer's CPU and initiates a confirmation protocol with the high assurance server. The confirmation protocol simply confirms that the TCBE is communicating with the high assurance server. The user can then select any trusted path option except login.

If the user chooses to switch session levels, then the current session encryption key is replaced by a new one from the high assurance server, which completed the processing required to support the user at the new session level. The TCBE resets the client host computer and overwrites its memory. Following the reset and clearing of volatile memory, the client operating system is booted on the host computer.

If the user chooses to return to the current session, the TCBE commands the host computer's CPU to resume operations.

When the user logs out of the session, the TCBE halts the host computer's CPU and overwrites its memory. Once the high assurance server has acknowledged receipt of the log out request, the TCBE clears its own memory including the session encryption key and the trusted path encryption key. The TCBE moves into a wait state until the SAK is pressed.

5. TCBE Physical Structure

As described in Functional Requirement 4, the TCBE will be placed on a separate controller card that is connected to the host computer. Since the TCBE needs access to a keyboard and a monitor for input and output, two methods of physically implementing the TCBE are considered.

a. Internal TCBE Controller Card

By placing the controller card inside the host computer connected to the bus via the motherboard, the TCBE has direct access to the host computer's memory, keyboard, and monitor. Acting as a device on the host computer's system bus, the TCBE only has to signal an interrupt to gain control of the host computer. However, the TCBE

must contend with the client operating system's control over the same memory, keyboard, and monitor. For example, in this configuration the TCBE must provide the keyboard interrupt functions for the host computer in order to check each keyboard scan code for the Secure Attention Key. In many cases it may not be possible to insert code that will provide TCBE control over those devices once the operating system is loaded.

b. External TCBE Controller Card

In this configuration, the TCBE would have its own keyboard and display device, (similar to a Microsoft Windows CE Personal Digital Assistant). The TCBE would not have to contend with what the client operating system does with the host computer's I/O devices. However, an additional communications layer must be added to the TCBE, in order for it to effectively control the host computer.

Both configurations have advantages and disadvantages. The design of the TCBE is general enough so that it could apply to either configuration. However, due to the lack of involvement of the client computer's operating system in implementing the TCBE as an external Controller Card, choosing the second configuration option for implementation is recommended.

B. OUTLINE OF THE BASIC MODULES

Chapter I provided the problem statement. Chapter II defined the requirements that the TCBE must meet in order to solve the problem. The analysis of the requirements determined the TCBE operational scenarios and states, which were described briefly in the TCBE Concept of Operations section of this chapter. Following the concept of operations, the major TCBE modules are produced. Each of the major modules may have

sub-modules that perform internal module functions, the additional modules are detailed in the Software Specification Document located in Appendix B.

1. TCBE Core

The first module is the TCBE Core. This module is first activated when the system is turned on and when there is a switch in the session levels. The module will initialize the other TCBE modules, format and overwrite the volatile secondary storage unit and hold the system in a secure waiting state until the Trusted Path to the high assurance server is invoked. Once the user has properly gained access to the system, this module will load or permit the client computer operating system to proceed and allow the host computer to communicate with the high assurance server. Object Reuse functionality is provided by this module's ability to overwrite the host computer's memory. This is the module where control of the host computer, the client operating system and the activation and deactivation of TCBE hardware reside.

2. Trusted Path Module

The Trusted Path is invoked when the user presses the SAK. The primary functions the Trusted Path Module is the establishment of a Trusted Path to the high assurance server, and to insure the user that the TCBE is connected to the high assurance server. The second function the Trusted Path Module provides is support for switching session levels. Logging out of the system is the third function that the module supports. This module contains the required protocols that initialize the connection to the high assurance server and that conduct the required handshaking between the TCBE and the high assurance server in order to correctly establish a Trusted Path.

The Trusted Path Interface Module lies within the Trusted Path. This module displays the front end of the Trusted Path to the user and allows the user to enter in Identification and Authentication information. The user will be able to enter in a User-ID and password in response to prompts issued by the high assurance server. The module will pass this information to the Trusted Path protocol, which will securely forward the information to the high assurance server. A token, such as a smart card or PCMCIA media, may also be used to provide additional authentication as well as provide encryption keys and algorithms to the TCBE. This module must have access to the monitor and keyboard for user interface functions.

3. Data Path Module

The Data Path Module serves as a pipe between the host computer and the secure Local Area Network. This means that the high assurance server appears to be nothing more than an additional storage device to the host computer. The module, via this representation, intercepts all user file accesses. The Data Path Module sends user data through the Encryption Module to the trusted server.

The Data Path Module is responsible for supporting the client end of the trusted computing base data transfer protocols (tcbdtp) when the TCBE is in the Data Path State. In the Data Path State, the TCBE provides the secure communications path to the high assurance server. When the user is finished with or saves the file, the TCBE supports the operation of those protocols that permit data to be stored on and retrieved from the high assurance server.

The tcbdtp are a set of protocols that may be similar to the standard file transfer protocol (ftp). The protocols' main purpose are to ensure that data travels correctly and securely between the high assurance server and the TCBE.

The Data Path Module may consist of a driver that is installed in the client operating system. This driver would provide the storage device representation of high assurance server to the operating system.

4. Keyboard Secure Attention Key Module

The Keyboard Secure Attention Key Module (SAK) provides the user the ability to invoke the trusted path. In the configuration where the TCBE is connected directly to the host computer's bus, this module must lie either logically or physically between the keyboard and the client operating system in order to ensure that it captures the appropriate key strokes when the user desires a trusted path. When the SAK is activated not only must the Trusted Path be initiated, but the client operating system must be halted and the TCBE prepared to overwrite the host computer's volatile memory.

5. Encryption Module

The Encryption module encrypts and decrypts TCBE data and user data. The module provides the management of the storage for a variety of encryption keys such as the Trusted Path Key and a session key. The Encryption module will lie between the communications device and the TCBE. By placing the encryption module in this position, the encryption module will provide a non-bypassable interface through which the services of the high assurance server are accessed. Hence, data cannot be passed to the local area network unencrypted. This module also provides the interface to a cryptographic device

module. The cryptographic device module can encapsulate hardware such as a FORTEZZA card or a software algorithm. This capability allows different encryption choices for each session or LAN configuration, and allows for easy upgrading of encryption devices, modules, and algorithms.

6. TCBE Hardware

The TCBE Hardware comprises the physical components of the TCBE. These components include the TCBE's microprocessor and memory, the network communications device and the cryptographic device (which could also be software). Additionally, the TCBE must handle a keyboard and a display device as well as the interface between the TCBE and the host computer. Depending on the configuration discussed in section A.5 of this chapter, the keyboard and display device may be the host computer's or separate components connected to a TCBE that resides outside of the host computer. In the case of the external TCBE, the TCBE to host computer interface would consist of a cable connection.

Each one of these devices will have their own TCBE drivers and interfaces to connect to the rest of the TCBE modules.

C. LAYER DESCRIPTIONS

The Trusted Computing Base Extension Layers are presented in Figure 1.Each layer will be described beginning with the layer that has the most privilege within the TCBE.

TCBE LAYER DIAGRAM

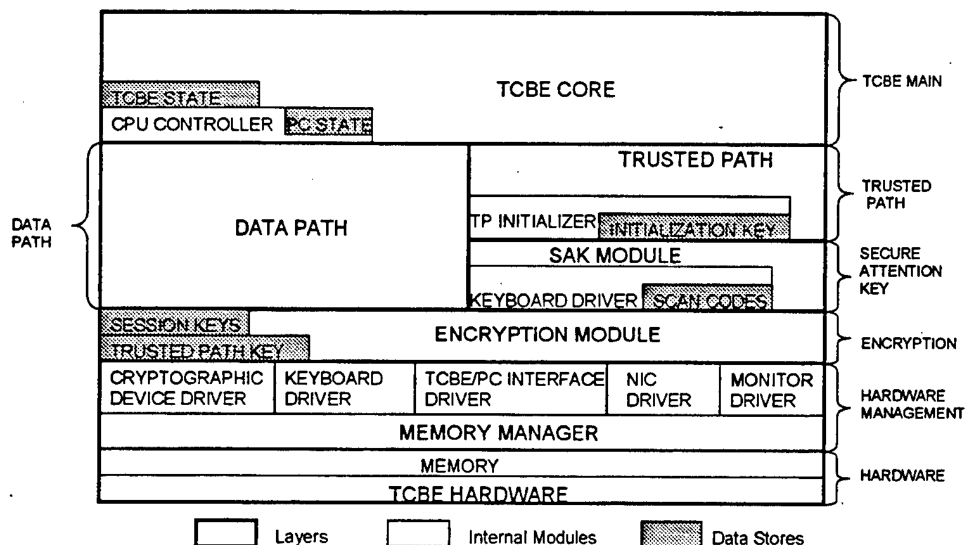


Figure 1 TCBE Layer Diagram

1. Hardware Layer

The TCBE Hardware layer consists of the TCBE Hardware as described in Section B.6. of this chapter. All other modules depend on this layer to perform their functions

2. Hardware Management layer

This layer includes the TCBE's Memory Manager and the device drivers associated with the each device.

3. Encryption Layer

The Encryption Layer consists of the Encryption Module. The Encryption Module relies on the both the cryptographic device and the network communications device in

order to perform its functions. The Encryption Module depends on the Hardware Management Layer.

4. SAK Layer

The next layer is the Secure Attention Key Layer. The Secure Attention depends on the keyboard to operate. The SAK requires direct access to the Hardware Management Layer and the keyboard's memory buffers.

5. Trusted Path Layer

The Trusted Path depends on the SAK to activate. It accesses the keyboard in order to receive user input and the monitor to display information to the user from the high assurance server. Since the SAK can be pressed at any time, the Trusted Path Module's keyboard access depends on the SAK module to capture all keyboard input. The Trusted Path module communicates to the HAS via the Encryption Module.

6. Data Path Layer

The Data Path Layer is the link between the host computer and the TCBE for passing user information. It communicates to the high assurance server via the Encryption Module. This module allows the host computer operating system applications to conduct normal operations such as preparing data for local storage before it is handed off to the Data Path Module for secure transmission.

7. TCBE Main Layer

The TCBE Main Layer consists of the TCBE Core Module and the modules necessary to initialize the TCBE and control the operation of the host computer. As such,

it depends upon the functions of the Trusted Path and the Hardware Management Layer to communicate to the host computer. It also depends on the Data Path Module when switching between Trusted Path and Data Path operating modes.

IV. SOFTWARE IMPLEMENTATION OF TCBE COMPONENTS

The software implementation of the TCBE is split into two portions. The first portion discusses the placement of the TCBE software within the client computer. The second part consists of designing the underlying operating system on the TCBE that allows the various modules of the TCBE to interact.

A. TCBE SOFTWARE PLACEMENT CONSIDERATIONS

Before implementing the TCBE's software, the location of the TCBE's code was determined. Placement of the TCBE code must satisfy the TCBE functional requirements discussed in Chapter II. In order for the code to be considered a part of the Trusted Computing Base it must demonstrate the characteristics of a reference monitor. Therefore, the placement of the code must ensure that the TCBE is always invoked and that untrusted mechanisms on the host computer can not circumvent it. Additionally, the TCBE must be tamperproof and its software small and modular to allow analysis of its self-protection features and correctness of operation. The architectural placement of the TCBE can affect the extent to which software mechanisms must be constructed to achieve the reference monitor objectives. In fact, incorrect placement of the TCBE can make it impossible to engineer a high assurance security architecture.

The three options considered for locating the code are: placing all of the code within a driver on the host computer's operating system, placing portions of the code in the host computer's memory alongside its operating system and some of the code on the TCBE hardware, or placing some of the code within the client's operating system and some on the TCBE hardware. The options are graphically represented in Figure 2.

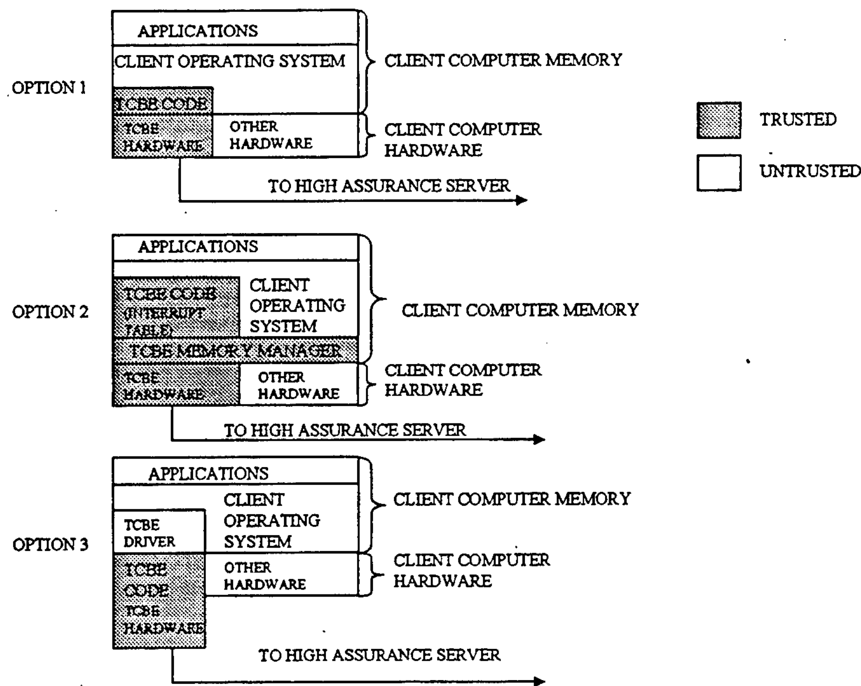


Figure 2 TCBE Software Implementation Options

1. TCBE Software Implementation Option 1

If the TCBE software were placed wholly on the client operating system, then the modules would be designed as file system and hardware drivers. The TCBE hardware would exist mainly as a network interface card. All authentication, encryption, and Trusted Path operations would run from the client operating system. By using this approach there would be smooth interaction between TCBE components and the client operating system. TCBE code would be able to take advantage of the client operating system's functionality as well as commercially produced software tools. Additionally it would be potentially easier for commercial software development firms to develop and produce the TCBE.

However, the TCBE software would need to be ported to each operating system. As the commercial vendors of operating systems upgraded or patched their software, the TCBE would have to be monitored to ensure that the improvements did not interfere with the drivers or break the TCBE's functional requirements. Since the TCBE is within the client operating system's structure it is also occupying the same domain as the operating system. This placement violates the fourth TCBE functional requirement, which states that the TCBE must maintain a domain for its own execution that protects it from external interference or tampering. If the commercial OS has a security flaw then the entire TCBE and by extension the secure network's security policy could be violated. This problem is mitigated somewhat by the fact that the OS could not be modified on the host computer, however it does not stop changes to the system during run-time, nor prevent malicious code passed by data driven attacks, from affecting the software during a session.

2. TCBE Software Implementation Option 2

The second option involves not placing any TCBE software within the client operating system at all. This could be implemented by placing the TCBE code with interface and host computer hardware control functions in the host computer's memory. The TCBE code would become a second interrupt vector table for the client computer. The interrupt vector table holds the addresses for host computer's interrupt handler code. The TCBE would "hook", i.e. replace, the interrupt addresses with vectors that pointed to TCBE interrupt handlers. All critical hardware and software interrupts managed by the original interrupt vector table would be handled by the TCBE software.

The TCBE hardware would consist of a controller card as well encryption devices. This option would capture all system inputs and outputs and direct then to the high assurance server without involving the client operating system, since the TCBE code has direct access to the host computer's hardware.

This option has several drawbacks. First, the code that is placed on the host computer and controls the hardware would be machine dependent, and therefore must be ported to each platform. The code must be small enough so that the client operating system and the TCBE can be active simultaneously. There is no guarantee that the client operating system will not overwrite the TCBE's portion of memory when it is initialized or during normal operations which violates TCBE Functional Requirement Four. In order to overcome this problem, a memory manager must be developed that partitions the host computer's memory. This partitioning system must not allow the client operating system to overwrite the TCBE's partition.

Since the TCBE code on the host computer would have direct access and control of the hardware it would be similar to constructing another small and primitive operating system on the computer. All interfaces, or drivers, to other devices (such as cryptographic peripherals) would have to be written at the hardware system level. This would involve writing the interfaces in the assembler code particular to the microprocessor of the host computer. Similar to the TCBE software, these device drivers would also have to be rewritten for each platform that the TCBE is installed in.

3. TCBE Software Implementation Option 3

The third option involves placing a few untrusted functions on the client operating system and having the TCBE software reside on the TCBE hardware. This implementation would present the TCBE solely as a network interface device to the client operating system. The interface would consist of a single untrusted TCBE "device" driver. The advantage of this alternative is that the device driver can use some of the features of the client operating system to perform simple non-critical tasks, such as data transmission preparation, while assuring that critical functions, such as encryption and authentication are located within the TCBE. The driver code would be the only software that would have to be changed to accommodate different operating systems, hardware platforms, or client operating system modifications. Additionally, the majority of the TCBE code need only be written once, and only has to perform the services outlined in the functional requirements.

The main drawback to this choice is that the TCBE cannot be a simple combination of a network interface card and an encryption device. The TCBE must have the ability to intelligently manage several different functions and hardware, as well as have enough memory to store data transmissions and TCBE code. Essentially the TCBE would almost have to be another computer. Therefore, in order to operate that computer, a small embedded operating system must be used for the TCBE. Because commercial embedded operating systems are proprietary, the source code cannot be accessed. This means that commercially developed embedded operating systems may have security flaws that are unacceptable for a secure environment; hence, a unique embedded operating

system may have to be developed for the TCBE. Additionally, commercial operating systems may not offer the specific functionality desired for the TCBE, or conversely, they may offer additional features that increase the complexity and the security vulnerabilities of the system.

Despite the complexity involved with the third option, it does provide the best means of ensuring a Trusted Computing Base Extension that meets the desired assurance requirements. The design of a small embedded operating system or executive for the TCBE Hardware is discussed in the next section.

B. TCBE OPERATING SYSTEM DESIGN

The Trusted Computing Base Extension will be designed as an executive. An executive is a system that does not require multitasking or dynamic rescheduling of processes. Since the software modules that run on an executive execute in a predetermined manner, an executive system can be less complex and more efficient than multitasking operating systems [4].

1. TCBE Executive Requirements

In order for the TCBE to function effectively and efficiently, the executive must perform a few operations reliably. First, the TCBE executive must be able to control and interface with its hardware. Next the executive has to manage memory for its code modules. The executive has to have a mechanism for defining a state. For the executive, a state is *defined* when it has its own or well-defined address space and stack configuration and is ready for execution. The executive will only start a state's operation when it receives the correct inputs and matches those inputs against a well-defined and

unchangeable set of starting criteria for each state. The executive will terminate a state's operations when the state has completed its functions or is interrupted by a set of predetermined events. The executive must correctly transition from one state to another. In order for the executive to transition between the states it must have a mechanism that correctly determines the correct state to transition to given the system's current state and a set of conditions.

Finally, the executive must have a method for handling asymmetric communications between states, the hardware, and the user. Asymmetric communications are inputs to the system that are not part of a planned sequence of occurrences that are encoded in a program. Examples of asymmetric communications are hardware and software interrupts and exceptions.

2. TCBE State Operations

In the Trusted Computing Base Extension software design, a state is defined as a common grouping of functions and values that describe and effect the TCBE's current operating behavior. For example, the Trusted Path State is the operation of the TCBE when it is communicating directly with the Trusted Computing Base on the High Assurance Server. It is made up of functions and variables that include the trusted path communications protocols, logging in and the switching of session levels. If the extension is not communicating with the TCB directly, then it is no longer in the Trusted Path State.

Each state may consist of several substates that further describe exactly what the TCBE is doing. Trusted Path Login is a substate of the Trusted Path state. However, functions that perform specific actions in support of a state are not substates. While the

TCBE may be in the process of encrypting data to go across the network, its mode of operation is not encrypting data. Instead, the data encryption function is used in support of a TCBE behavior such as logging in, which require the encryption of user information.

A state machine engine is represented by a table of transitions and actions [5]. In the TCBE executive, the state matrix consists of the states of the TCBE and a set of actions that include asymmetric communications, called events, which cause transitions. Figure 3 represents a state machine engine.

	STATES			
EVENTS	1	2	3	4
A	2	4	4	4
B	1	3	4	2
C	1	3	3	3
D	3	4	3	4

Figure 3 State Machine Engine

The state machine engine constitutes a simple algorithm that keeps track of the present state of the system in a current state table and monitors actions with an event table. When an event occurs, the engine checks the state matrix for the position within the matrix of the event and the current state and then transitions according to what is at that location in the matrix.

For example, from Figure 3, if the system is currently in state 3, and event A happens then the executive will transition to state 4. If event C happens and the system is in state 3, then no transition will occur. Additionally, all events are serialized so there is no conflict caused by simultaneous events.

a. TCBE Software States Defined

The analysis of the concept of operations, and the TCBE architecture presented in Chapter III provided the basis for determining the states that form the TCBE's operations. The TCBE operates in four main software states. The main states diagram is displayed in Figure 4 below.

The TCBE system starts from an off or inactive state. Since, there is no power to either the TCBE or the host computer, the TCBE Off state is not considered an operating state. The next state is the Trusted Computing Base Extension Initialization State (TCBE Init). In this state the TCBE's software is loaded and its hardware is

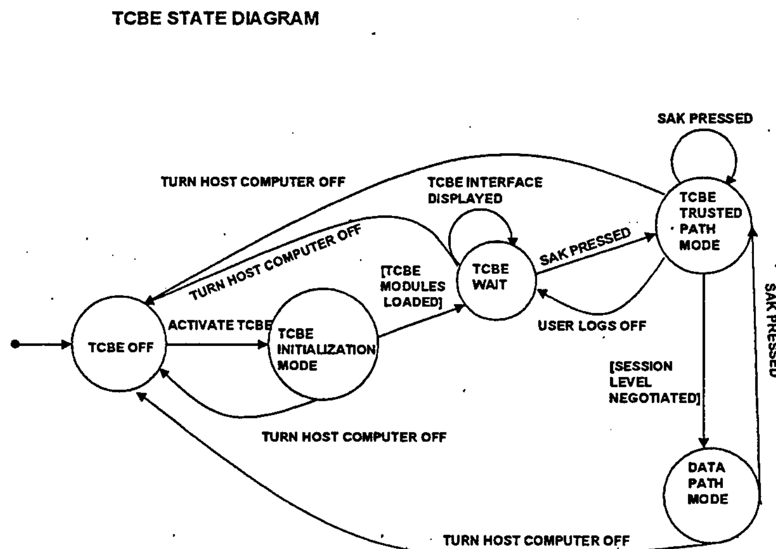


Figure 4 TCBE State Diagram

initialized. The host computer is also turned on in this state, although its operating system is not yet loaded. This state automatically transitions to the next state which is the Trusted Computing Base Extension Wait (TCBE Wait). In this state the TCBE is waiting for the Secure Attention Key to be pressed in order to invoke the Trusted Path. TCBE Wait is only a waiting state.

Once the SAK is pressed the TCBE transitions to the Trusted Path State. This state is made up of several substates. The substates are presented in Figure 5. The first substate is the Trusted Path Initialization State (TP Init). It is in this state that the

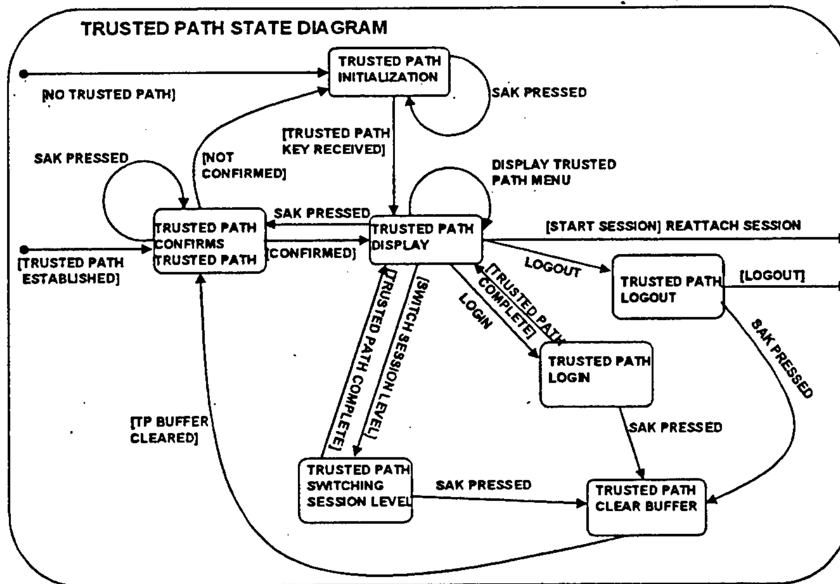


Figure 5 Trusted Path State Diagram

Trusted Path is first set up between the user and the Trusted Computing Base on the high assurance server for a particular session. The next substate is the Trusted Path

Confirmation State (TP Confirm). In this state, the user, via the TCBE, verifies that a Trusted Path exists between the TCBE and TCB. The next substate is the Trusted Path Display (TP Display). In the Trusted Path Display the user selects the Trusted Path options of login, logout, switch sessions, or reattach to a session. Because of the behavior of the TCBE during these options, login (TP Login), logout (TP Logout), and switch sessions (TP Switch Session) are also considered substates of the Trusted Path. Reattaching to a current session is merely a transition function between the Trusted Path State and the last state, the Data Path State.

The Data Path State is broken into two substates that are presented in Figure 6. The first substate is the Data Path Initialization State (DP Init). In this state the TCBE loads the client computer's operating system into memory. Once this operation is complete, the TCBE transitions to the Data Path Active (DP Active) state. This state is characterized by use of the client computer as a normal workstation with the secure flow

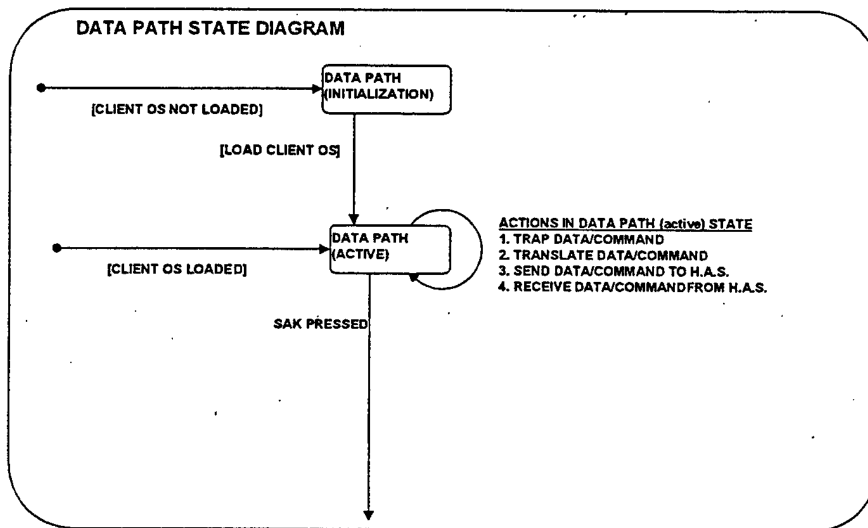


Figure 6 Data Path State Diagram

of user application data between the host computer and the high assurance server.

b. TCBE Events Defined

Trusted Computing Base Extension Events are asymmetric functions that effect a transition between states. The most critical event is the Secure Attention Key (SAK). When the SAK is pressed, the TCBE must conduct the actions that will result in a transition to the correct state. The SAK event incorporates the use of the keyboard and its associated interrupts.

The next event is the transition event. When a state has properly completed its actions or certain conditions are met (signified by [] on the state diagrams), the state activates a transition event, which causes the TCBE to shift to the next state. Specialized transition functions include the login, logout, switch session, and reattach session events. These events are only called from the Trusted Path Display State and cause a transition from that state to the state corresponding with the event.

The final event is the keyboard event. This event is caused from the keyboard hardware interrupts. It only affects the TCBE in two states. In the TP Display state, the keys entered from the keyboard allow the user to choose Trusted Path options such as login or logout. In the TP Login State, the keyboard is used to enter user identification and authentication information. In both of these states, the keyboard event causes no transition, it only allows the functions within those states to read and if necessary, display, the keyboard scan codes. Otherwise the keyboard is allowed to function without affecting TCBE operations.

c. *TCBE State Matrix*

The state matrix for the TCBE is presented in Figure 7. The states are presented across the top of the matrix with the events listed along the left side. Each state is assigned a number which is placed in the appropriate position within the state matrix. Blank locations signify that no action is taken when the two conditions that form that point exist.

TCBE STATE MATRIX

EVENTS \ STATES	TCBE INIT	TCBE WAIT	TRUSTED PATH	TRUSTED PATH INIT	TRUSTED PATH CONFIRM	TRUSTED PATH DISPLAY	LOGIN	LOGOUT	SWITCH SESSION	DATA PATH INIT	DATA PATH ACTIVE
	1	2	3	4	5	6	7	8	9	10	11
SAK	2	3	4	4	5	5	5	5	5	5	5
TRANSIT				6	6		10	2	10	11	
REATTACH						11					
KEYBOARD						6	7				
LOGIN						7					
LOGOUT						8					
SWITCH						9					

Figure 7 TCBE State Matrix

d. *TCBE Executive Operation*

TCBE Executive operation is presented in the following Pseudo-code.

Initialize System
    Clear memory
    Initialize Data Structures

Define States
Assign address space
Assign Stack space

Build State Table
Set Initial State and event
Call State Machine Algorithm
Do {State transitions until TCBE is turned off
Events call new state transitions}

States are defined from functions that are written in the program code. The definition of the state includes the function name, its parameters, and a state identification number. Each function is assigned a set address space and fixed amount of memory.

The state table is built by assigning each location in the state matrix a pointer to the state identified by its state identification number. The executive then assigns an initial state and a transition event. The State Machine Algorithm is then called and continues until the TCBE is turned off.

e. State Machine Algorithm

The State Machine Algorithm is presented in the following pseudo-code.

Machine called
Save current State
Select the next State
Check Event Table for entry
Check Current State Table
Get new state from location in the State Table
Reassign memory and reinitialize TP Init, TP Confirm, and Login if required
Switch Control to new State
New state operates until next event

Certain states such as the TP Confirmation state and the Login state need to have local variables cleared and their entire process restarted after events to prevent user and trusted path authentication data from being leaked. If a user is logging in and has entered user identification data but has not finished before being interrupted and pressing the Secure Attention Key, then that information may still reside in local storage on the TCBE. Another user may be able to manipulate the system and get that information if it is not removed. Therefore those states are restarted from the beginning of their instruction sequences.

The Data Path State will start at the next instruction in its instruction sequence if its session is interrupted and then reattached to by the user. If a different session level is selected, or a new user logs in, then the Data Path is restarted from the beginning of its instruction sequence.

V. HARDWARE IMPLEMENTATION OF TCBE COMPONENTS

The hardware of the Trusted Computing Base Extension is intended to maximize the functionality of the system. The hardware configuration of both the TCBE and the host computer are important to ensuring that Functional Requirement Four, which calls for the TCBE to operate a domain separate from the that of the host computer, and Functional Requirement One, which requires that there be object reuse and data remanence services at the client computer, are satisfied.

A. TCBE HARDWARE REQUIREMENTS

Chapter III discussed two options for the physical placement of the Trusted Computing Base Extension Hardware. Whether the TCBE is implemented as a card that is mounted inside the host computer or as a separate terminal device with a communications interface with the host computer, the overall design requirements remain the same.

1. Memory

The allocation and use of memory is the most important part of the TCBE. The system's memory will contain not only the operating software but it will also contain TCBE databases such as encryption keys, which include the session keys and initialization keys. The operating software should reside in read only memory (ROM). Once the operating code is "burned" into the ROM it cannot be changed. This is the equivalent of designing a microprocessor as the TCBE's state machine engine. The only way that this code can be changed is if the memory is replaced. Replacement of the TCBE's hardware

should only be done under controlled conditions. Checksums can be incorporated into the operating code. Checksums use a memory and hardware signature to verify that the code has not been changed.. If the hardware has been changed or removed there will be no method of communicating with the Trusted Computing Base at the server.

The TCBE must have volatile memory that stores TCBE databases, TCBE code module execution stacks, user access information, as well as user interface data. The TCBE's software design includes specific interfaces between modules and layers that prevent overflows of data between the various functions of the TCBE. Additionally, the design of the executive uses a memory allocation system that provides each state of the TCBE its own block of contiguous memory within the system's random access memory upon initialization. The memory allocation system also includes functions to reinitialize the memory for each state upon a state transition if required. The state machine algorithm, presented in Chapter IV section B.2.e, determines which states get reinitialized when a state transition occurs.

The TCBE should also have procedures to completely clear the TCBE's random access memory of encryption keys and other session identification and interface information when a session is complete. The TCBE operating system includes procedures for erasing the areas of volatile memory used by each state when the system transitions between states, this function could include the same algorithms that are developed to clear the host computer's memory. Finally, when the TCBE loses power, the TCBE's entire random access memory must be cleared.

2. Microprocessor

The TCBE's microprocessor plays a key role. As described in Chapter IV, the TCBE system must be able to handle a variety of functions efficiently. There is no requirement for multitasking of TCBE functions. Multitasking of the defined TCBE functions will lead to a more complicated system that will be harder to verify that it is operating correctly. The Trusted Computing Base Extension's CPU should be sufficiently fast enough to make TCBE operations transparent to the user. The central processing unit should have sufficient power and speed to efficiently handle encryption algorithms that may be software implementations within the TCBE.

3. Communications Interfaces

The TCBE controls two communications interfaces. The first interface is the connection to the host computer. The TCBE virtualizes the network interface card (NIC) to the host computer via the TCBE driver so that the host computer thinks it is communicating with a regular NIC. The TCBE ensures the proper encryption and transmission of the data sent to the high assurance server. The bandwidth of the connection between the TCBE and the host computer should be large enough that routine passing of data and commands is transparent to the user.

The second interface is the one established between the TCBE and the TCB over the local area network. This connection will be through a common network interface card. The NIC operations should not be modified in any way. All cryptographic functions should take place in the Encryption Module, which includes encryption devices and software, on the TCBE. The NIC only passes encrypted data to and from the Trusted

Computing Base Extension. This separation of functions allows troubleshooting of the NIC and its communications protocol without affecting the TCBE's cryptographic functions. In addition, it permits modularity; the cryptographic and trusted path protocols that are established between the TCB and the TCBE may be examined separately.

B. HOST COMPUTER HARDWARE REQUIREMENTS

The host computer requires few changes from a normal workstation or personal computer configuration. The major changes are interrelated and include the amount of random access memory available to the system, and the presence of hard disk drives and other secondary storage devices.

1. Hard Disk Drive Presence

A hard disk drive is used as permanent storage of files and data on workstations. It normally consists of metallic platters with a magnetic coating that holds the information even if there is no power. Because it is non-volatile storage, many commercial operating systems rely on the hard disk drive for loading operating system software into the workstation's memory, and storing user information, operating system configuration data, and page-swapping of data between real and virtual memory.

The TCBE is required by Functional Requirement One to prevent object reuse and data remanence at the client workstation. Moreover, the requirement states that operating system software and user applications will be stored on read-only media. Therefore if the system were to have a hard drive, its capability to write data would have to be disabled after the operating system and applications are loaded. This would require modification of

the COTS operating system, which is not possible due to the proprietary nature of commercial software applications. In order to accomplish read-only operations, the operating system must be configured at this point to not swap pages of information between memory and the hard drive. If the host computer has sufficient random access memory then the impact of the lack of virtual memory on a hard drive for paging is minimal.

There are two alternatives to the hard drive in the host system. The first alternative is to use a compact disk – read only memory (CD-ROM) as the repository for the operating system and applications. This method would allow the system security administrator to record system disks with the correct configuration data as well as the proper applications. Once these disks are recorded they cannot be overwritten during normal operations.

A potential vulnerability to this approach is that newer CD-ROM systems permit recording on unused portions of the CD. This would require that the CD-ROM be initialized as a "multi-session", which allows multiple recordings on a single disk, by the original recorder of the disk and the presence of CD-ROM recording software installed in the operating system. System configuration can be used to prevent this vulnerability. The system administrator should not initialize the CD-ROM as "multi-session" nor should the recording software be installed in the original operating system configuration for each workstation.

The host computer would require a CD-ROM drive. This should not be a problem since most computers and workstations come with these drives already installed. The main disadvantage to using CD-ROM disks is that their data transfer rates are

considerably slower than those of hard disk drives. This may cause an unacceptable delay between sessions, as the entire operating system must be reloaded each time a new session is started or when there is a switch between session levels. Additionally, each workstation's CD-ROM would have to be replaced whenever the system administrator decided to update or add software to the secure LAN's client computers.

The other alternative is to store the operating system and workstation applications on the high assurance server and have the server send a copy of the software to the client computer at each login or change of session levels. In this case, the high assurance server ensures the integrity of the software and this makes configuration management of client computer software easier as all clients use copies of the one centrally managed software component. Additionally, this option may be less expensive than using CD-ROMs, since any change in that first alternative's operating system or application software requires the purchase new CD-ROMS for each client computer on the network.

There is one main disadvantage to this alternative. This drawback is the size of current commercial operating systems and applications. A basic configuration of Microsoft Windows NT 4.0 requires 80 Mbytes of hard disk drive space. This does not include commonly used Microsoft Office applications. On a 10 Mb/sec local area network, it would take approximately sixty-four seconds to transmit the operating system code, if the network was not busy. This does not include the time for the client system to load the system software in its memory. This problem may be alleviated by faster networks and data transfer protocols, but as the size of commercial software and the number of clients on the network grows, this wait time may become unacceptable.

If the host computer cannot have a hard drive it must not have any other secondary storage devices including floppy disk drives. If a floppy drive is associated with the system it should be disabled or removed. The high assurance server should include services for transferring data to secondary storage devices that it directly controls.

2. Amount of Random Access Memory

The second major change required to turn a workstation into a host computer for the TCBE relates to the amount of random access memory at the workstation. A minimum of 128 Mbytes of random access memory should be installed on the host computer, with 256 Mbytes considered optimal for today's commercial software. Since RAM is easy to wipe clear after each use, the host computer will use RAM exclusively to run its operating systems and applications and locally store user produced data. Additionally, the operating system may need to use RAM as virtual memory to write pages to during page swapping operations since the secondary storage device normally used for that function would be inaccessible.

The introduction of computers such as Network Computers and Windows terminal devices may provide may provide the optimum client for the secure network. These computers are designed to operate with random access memory only and without the use of non-volatile secondary storage such as hard disk drives or floppy disk drives [6].

C. IMPLEMENTATION OF TCBE

A subset of the Trusted Computing Base Extension is implemented for this thesis. The subset is designed to show that the state machine engine operates properly when the Secure Attention Key is pressed.

The TCBE subset is implemented on an Intel 80486 based personal computer. The microprocessor operates at 33 MHz. The system has eight megabytes of random access memory and a network interface card, as well as a keyboard, monitor, and a floppy disk drive. This implementation resembles the TCBE option of placing the TCBE in a separate terminal device such as a PDA. In fact, the personal computer used can be thought of as a trusted PDA with a NIC attached.

The TCBE executive was written in assembly and the C programming language using the Borland C++ (v 5.00) compiler and Turbo Linker. This program is approximately 3000 lines of code in length. For this implementation, the executive relies only on the personal computer's basic input-output system (BIOS) for access to the system's hardware. It is loaded into the personal computer using the system's disk operating system (DOS). Once loaded, the TCBE calls no other DOS functions, including system calls and interrupts. This version of the implementation used less than 70000 KB of the personal computer's RAM. Future implementations using a compiler and linker optimized for a controller card or PDA platform will most likely use less space. Ultimately, the system DOS will have to be replaced to provide a complete assurance architecture.

The executive included the capability for the Secure Attention Key, alt-S, operations to be examined using a test function. As proof of concept each of the TCBE's states was implemented as stub procedures that displayed their state name when they were activated. Appendix C contains the source code for the executive and the make file and commands required to load and use the software. The stub procedures presented in the

source code provide the examples necessary to complete the functions of the TCBE so that they interface correctly with the TCBE executive.

VI. CONCLUSIONS

The design of a Trusted Computing Base Extension is feasible. None of the elements that went into the design of the TCBE are beyond current technological capabilities. This chapter discusses the cost effectiveness of the TCBE as well as recommendations for future work required in order to make the TCBE function as part of a secure local area network.

A. COST EFFECTIVENESS

Construction of the TCBE as a combination controller card and network interface card or as a personal digital assistant type device would cost less than $500. The additional costs to the system would involve the addition of system memory. Given the current prices for random access memory, 256 Mbytes of RAM should cost less than $200. Therefore total additional cost should be approximately $700 per system or less if the TCBE is produced in large quantities. This compares to the approximate $4,500 cost of having three separate computers for each security level, or costs an order of magnitude higher for purchasing an evaluated high assurance multilevel system for each user on the local area network. If intelligence information is being processed, the number of security levels increases dramatically and so the savings are considerable.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

1. Host Computer

Host computer research can be broken into three areas. The first area is host computer operating system control and loading. Chapter V discussed the issues

surrounding the loading, or booting, of the client operating system. Control of the operating system can be accomplished by installing the TCBE via the device driver on the host computer. The device driver will be written such that when the device interrupts, the operating system halts and the host computer's microprocessor follows the TCBE's instructions.

Since it is possible for the host computer's microprocessor to save the state of the CPU when it is interrupted, it may be possible to save the state of the client operating system when there is a session level switch [7]. This would allow only the restarting of the state of the session operating system as opposed to the current mode where the entire client operating system must be rebooted. A method to store and reload the operating system state must be developed.

The second area of research involves implementation of the TCBE's object reuse functions. The TCBE must support object reuse of host computer memory, which includes clearing all memory including caches within the host computer. The procedures to conduct the clearing functions performed by the TCBE must be developed. Present designs only provide the program interfaces for those functions.

The third area of research for the host computer involves the structure of the system's random access memory. Optimally the host computer's RAM should be structured into logical RAM drives. There should be a minimum of two RAM drives built when the host computer is turned on. The first RAM drive is where the client operating system is loaded. The second RAM drive is where the client OS would temporarily store user-produced data. This would simplify the configuration for the client operating system. Each RAM drive would appear to the OS as a physical drive such as C: and D: drives.

There is source code available for constructing a RAM drive and an associated file system [8]. Research needs to be conducted into constructing two separate drives from one source of physical memory and determining how to construct the RAM drives from functions resident on the TCBE.

2. Trusted Path Research

The establishment and maintenance of the Trusted Path between the TCBE and the TCB is perhaps the most important component of the secure local area network. If the Trusted Path is not constructed properly, then intruders on the net can compromise the entire system by either impersonating the High Assurance Server or authorized users. The design of the TCBE places the Trusted Path operations into five tentative states: Initialization, Confirmation, Login, Logging Out, and Switching session levels. Of those five states, the Initialization State is the most critical phase of the establishment of the Trusted Path.

In the Initialization State, the TCBE must first securely identify itself to the Trusted Computing Base then the TCB must authenticate itself to the TCBE. The first question to be solved is the identification of the TCBE to the TCB. If there is a permanent identification code stored with each TCBE, then a malicious user could place the TCBE in a host computer that has local secondary storage capability or copy the code onto some non-TCBE device.

Once a method for TCBE identification is developed, there must be a way of protecting the secrecy and integrity of the identity information for transmission over the LAN. The techniques for protecting this information may include software, hardware, and

configuration management equipment and procedures. One issue involved in developing these protection features is the use of encryption. Any encryption algorithm used to support these techniques must allow decryption by the TCB, so it would have to be previously negotiated. If the algorithm is negotiated beforehand, then it must be decided where the keys for the encryption scheme are stored. This question leads to determining the best and most secure way to store the key(s), which is similar to the storage of the TCBE identification code. Fortunately, there has been research conducted that looks into the requirement for security in a distributed system and several protocols such as Needham-Schroeder [9] and Kerberos [10] have been developed. But none of the research or the developed protocols seems to answer the requirements for the Trusted Path. The remaining phases of the Trusted Path development are solvable once the Initialization protocol is developed.

C. CONCLUSIONS

The Trusted Computing Base Extension provides an economical method to connect commercial off-the-shelf personal computers or workstations to a secure local area network. A systems software engineering approach was taken to produce a design for the TCBE that satisfies the security requirements of connecting to a multilevel system while requiring minimum changes to a COTS computer or operating system and application software.

Chapter II listed the functional requirements for the Trusted Computing Base Extension. These requirements should serve as guide for development of future versions of the TCBE or other devices that are designed to enhance the security of a system or

network. Chapter III provides the analysis of the requirements that went into the design of the system. The concept of operations describes the operations of the TCBE as a sequential process engine that is to be encoded in the TCBE operating code and applications. Chapter IV detailed the development of the software needed for the TCBE. The design of this system requires a clear understanding of the hardware of both the host computer and the physical components of the TCBE. The outcome of this design is a limited operating system, or executive, that can be evaluated for its correct operation. The TCBE operating system can be expanded, once it is shown to be operating correctly, to include additional states, or functions, as the need arises.

Chapter V discussed the hardware requirements for the client computer and the TCBE. None of the hardware needs to be specifically developed for this system. Client computers are easily available. The components that make up the TCBE can be manufactured by an embedded controller card producer or purchased as part of a modified Personal Digital Assistant. In the future, thin client systems such as Network Computers, may be a viable option for the secure local area network.

APPENDIX A. TCBE DESIGN DOCUMENTATION

The appendix represents a preliminary design for the TCBE. Additional effort will be required to ensure that the layering of the TCBE meets the requirements for a high assurance architecture.

The documents presented here follow the object-oriented methodology developed by Rumbaugh, et al.[5]. The requirements for the TCBE were discussed in Chapter II. These requirements formed the basis for the TCBE design. The scenarios of the TCBE were produced first. The scenarios are a step by step presentation of the expected operation of the TCBE from turning it on to off.

From the scenarios, state diagrams were developed. These state diagrams graphically represent the behavior of the system at specific points in its operation. The state diagrams were the basis for developing the layers and the dependencies of each of the code modules and the entire TCBE. The state diagrams are also used to develop the TCBE executive.

Following from the reference, the object modules were defined. They are presented in this document in graphical and tabular format. An object diagram was drawn that detailed the function of the software modules. The combination of the state diagrams, layer and module functions, and object diagram were combined with a hierarchical, top-down approach to system architecture. This permitted objects to be placed in partially ordered layers such that dependencies were well understood and circularities eliminated from the software architecture. The result was the System Software Specification Document that is presented in Appendix B.

A. TCBE SCENARIOS

1. Initialization

A.

1. Turn on the host computer (if the TCBE is installed within the host computer then it turns on when it receives power)
   a. BIOS POST Test
   b. The TCBE begins functioning in place of the client operating system, since there is no secondary storage holding the client's operating system.
   c. TCBE Core Loaded.
   d. TCBE in Initialization Mode.
   e. TCBE Core initializes all devices.

2. Turn on the host computer; Turn on the TCBE (if TCBE is a separate device)
   a. BIOS POST Test.
   b. Host computer start up stopped here since there is no operating system installed on secondary storage.
   c. TCBE Core Loaded.
   d. TCBE in Initialization Mode.
   e. TCBE Core initializes all devices.

B. TCBE Core Overwrites host computer memory
C. TCBE Core halts host computer CPU
D. TCBE Core loads all TCBE Modules
E. TCBE Interface Displayed
   1. TCBE Video Screen Cleared
   2. TCBE Logo displayed

2. Secure Attention Key (SAK) pressed on TCBE keyboard

A. Keyboard Generates SAK interrupt.
B. SAK Module sends activate Trusted Path command to TCBE Core.
C. TCBE Mode switched to Trusted Path (TP) Mode
D. Encryption module mode switched to Initialization by TP Module
   1. Session Key Store Flags set to Null
   2. Trusted Path Key Store Flag set to Null
E. Trusted Path conducts initialization sequence with the HIGH ASSURANCE SERVER; to ensure that the appropriate computers are talking to each other.

F. Trusted Path module receives Trusted Path encryption key (TPK) from High Assurance Server.
   1. Trusted path Key Store Flag set to active
G. Trusted Path Established

H. Go to **Trusted Path Interface**

3. Trusted Path Interface

A. Trusted Path Interface presented on TCBE Monitor on command from the high assurance server
B. User enters one of the following options 1. Login; 2. Logout; 3. Change Session Level; 4. Start Session; 5. Reattach Session
C. If Login go to **Trusted Path Login**
D. If Start Session then go to **Data Path Mode**
E. If Logout go to **User Logs Off**
F. If Change Session Level go to **User switches session level**
G. If Reattach Session go to **User reattaches to current session**

4. Trusted Path Login

A. Trusted Path login interface presented on TCBE monitor on command from the high assurance server.
B. User enters login name and password
C. User info encrypted with TPK goes through TP Protocol to High Assurance Server
D. User enters desired session level (make sure there is a default session level at TCBE)
E. Session Level info sent to High Assurance Server
F. High Assurance Server sets a session level at desired Session Level, returns a session key to Trusted Path Module
G. Trusted Path sends key and session level to Encryption Module
H. Encryption Module receives Session Key, stores it under a session level key store
   1. Session Key Store Flag set to active
I. Encryption Module tells TP Module that it has received the Session Key
J. TP Module switches Encryption Module to the appropriate session mode
K. Trusted Path Module tells TCBE Core that session is ready
M. Go to **Trusted Path Interface**

5. Data Path Mode

A. TCBE State switched to Data Path Mode

B. If Client OS is not loaded then go to <u>Data Path Mode (Initial)</u>
C. If Client OS is loaded then go to <u>Data Path Mode (Active)</u>

1. Data Path Mode (Initial)

(If initial session)

1. TCBE Core tells CPU to resume
2. TCBE Core Overwrites host computer RAM.
3. TCBE Core constructs memory resident file system;
4. TCBE Core loads client OS in File System
5. TCBE core loads Input/Output (I/O) Trap;
    a. IO Trap stays resident and communicates with NIC via encryption module
6. Client OS completes loading
7. Go to ***Data Path Mode (Active)***

2. Data Path Mode (Active)

1. If the host computer CPU is idle then the TCBE Core tells CPU to resume.
   NIC accepts Data.
2. User operates on top of client OS
3. User I/O trapped by I/O Trap:
    a. User command translated to High Assurance Server command
    b. TCBE data block constructed with command
    c. Data block encrypted w/session key
        1. Data block sent to Encryption device
        2. Data block encrypted
        3. Data block returned to the Encryption Module
    d. Data block sent to High Assurance Server
    e. Data block decrypted by High Assurance Server
    f. High Assurance Server processes request
    g. High Assurance Server constructs data block
    h. High Assurance Server Encrypts data block
    i. High Assurance Server sends data to TCBE client
    j. Communications Device receives data block
    k. Send data block to encryption module for decryption (similar to c. Data block encrypted w/session key)
    l. Decrypted data sent to Data Path
    m. Data Path sends data to client OS

4. OS I/O not trapped sent to secondary memory / or controlled path

3. User Invokes Trusted Path While Using Client OS

1. SAK Pressed
2. SAK sends signal to TCBE Core
3. TCBE State switched to Trusted Path; TCBE core idles client OS/Host Computer; Network Interface Card stops accepting data
4. Encryption Module Switched to Trusted Path mode
5. TCBE calls TP Module
   a. Path confirmed
   b. TP Module controls keyboard; monitors for SAK
6. Go to ***Trusted Path Interface***

6. User Logs Off

A. Go to ***User Invokes Trusted Path While Using Client OS***
B. User selects logout
C. TCBE Core halts CPU
D. Logout signal sent to high assurance server
E. TCBE Erases Client PC memory; Session Keys; Trusted Path Key;
F. Logout complete signal received from the high assurance server
G. TCBE interface displayed
H. Host computer Screen blank

7. User Switches Session Level

A. Go to ***User Invokes Trusted Path While Using Client OS***
B. User selects switch session level
C. User enters desired session level
D. Session level information sent to high assurance server (encrypted)
E. High assurance server sets session at desired session level returns a session key to Trusted Path Module
F. Trusted Path sends key and session level to Encryption Module
G. Encryption Module receives Session Key, stores it under session level key store
   1. Session Key Store Flag set to active
H. Encryption Module tells TP Module that it has received the Session Key
I. TP Module sets Encryption Module to that session mode (encryption module knows know to use the key that matches the mode until told by the TP Module to switch)
J. TCBE erases secondary memory (Or stores this information on the board if possible)
K. TCBE prior session key (Or stores this information if possible)

L. Encryption Module tells TP module that it has received the Session Key

M. Trusted Path Module tells TCBE Core Trusted Path complete

N. Go to **Trusted Path Interface**

8. User Reattaches To Current Session

A. TP Module sets Encryption module to the session mode active before SAK pressed.

B. Trusted Path Module tells TCBE Core Trusted Path complete.

C. Go to **Data Path Mode**

9. User Presses SAK While In Trusted Path Mode

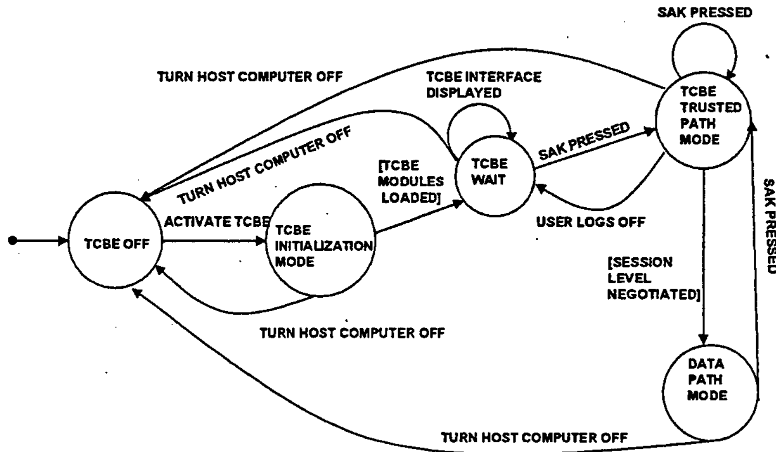
A. TP module does the following

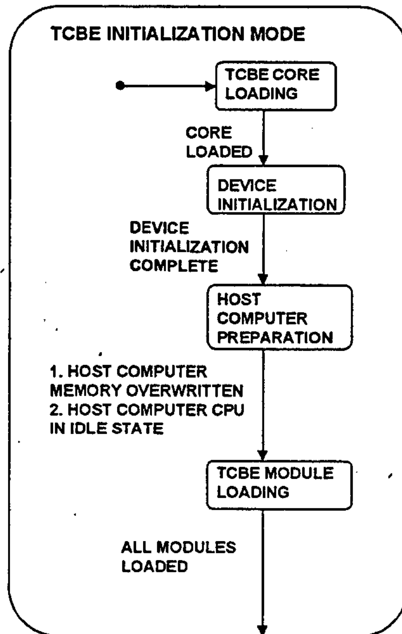
1. If confirming Trusted Path then restart confirmation process
2. If initializing Trusted Path then restart initialization process
3. If Trusted Path is idle then confirm Trusted Path
4. If switching sessions, logging in or out then do the next three steps:
   - a. Trusted Path Buffers/Memory cleared
   - b. Trusted Path Confirmation Protocol Initiated
   - c. Trusted Path confirmed

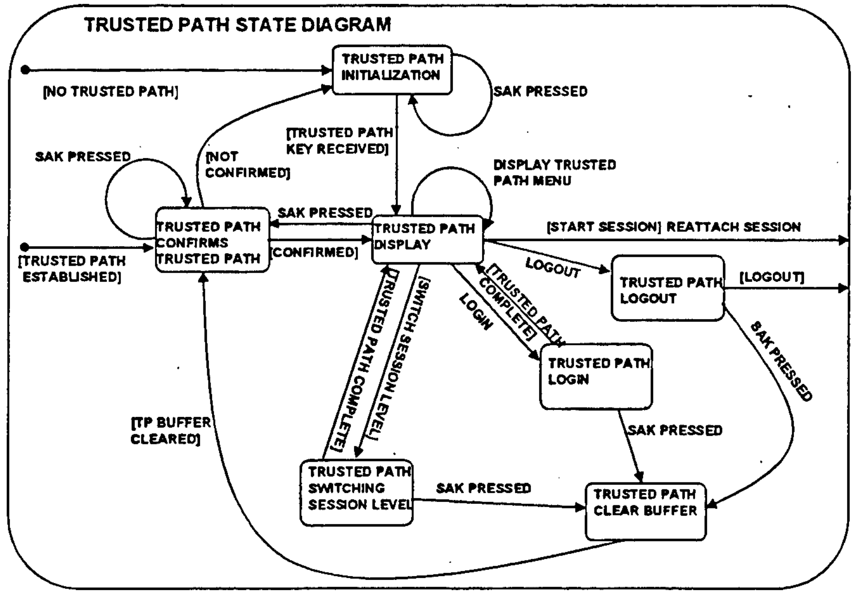
B. Go to **Trusted Path Interface**

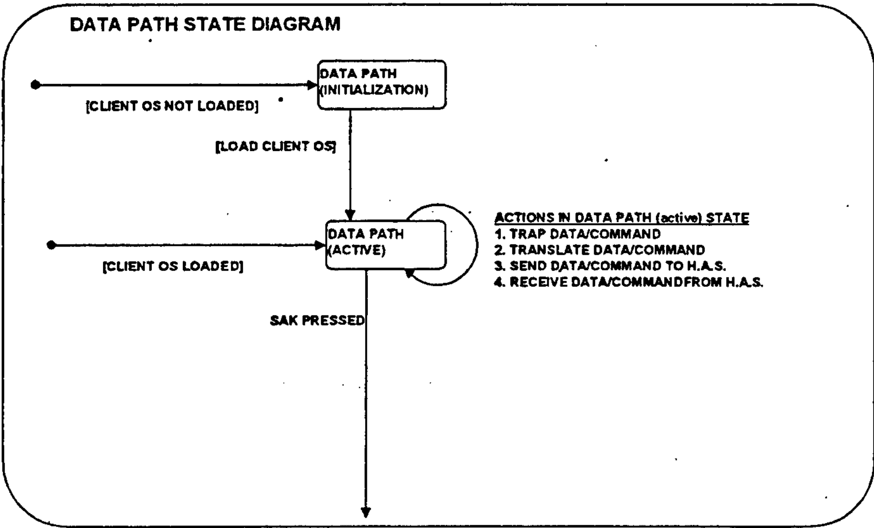
B. TCBE STATE DIAGRAMS

TCBE STATE DIAGRAM

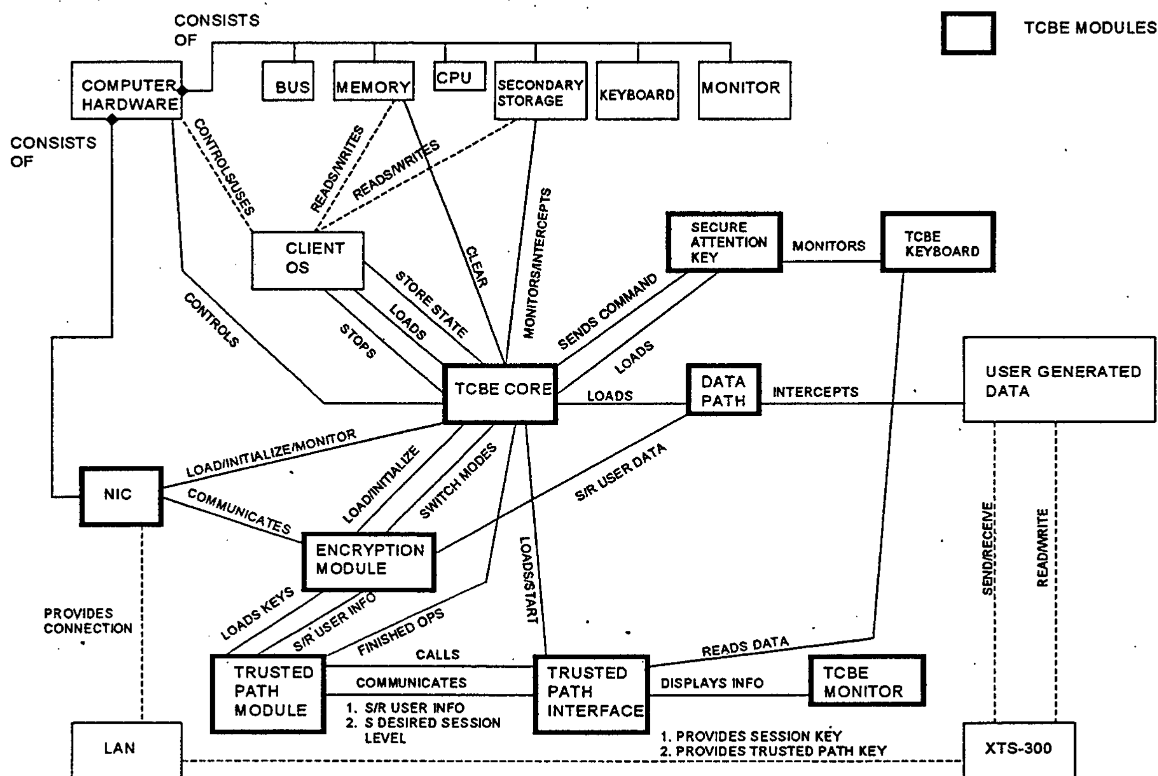





TRUSTED PATH STATE DIAGRAM
TRUSTED PATH INITIALIZATION
[NO TRUSTED PATH]
SAK PRESSED
[TRUSTED PATH KEY RECEIVED]
SAK PRESSED
[NOT CONFIRMED]
DISPLAY TRUSTED PATH MENU
SAK PRESSED
TRUSTED PATH CONFIRMS TRUSTED PATH
TRUSTED PATH DISPLAY
[START SESSION] REATTACH SESSION
[TRUSTED PATH ESTABLISHED]
[CONFIRMED]
LOGOUT
TRUSTED PATH LOGOUT
[LOGOUT]
[TRUSTED PATH COMPLETE]
LOGIN
[SWITCH SESSION LEVEL]
[TRUSTED PATH COMPLETE]
SAK PRESSED
TRUSTED PATH LOGIN
[TP BUFFER CLEARED]
SAK PRESSED
TRUSTED PATH SWITCHING SESSION LEVEL
SAK PRESSED
TRUSTED PATH CLEAR BUFFER


DATA PATH STATE DIAGRAM
DATA PATH (INITIALIZATION)
[CLIENT OS NOT LOADED]
[LOAD CLIENT OS]
DATA PATH (ACTIVE)
[CLIENT OS LOADED]
ACTIONS IN DATA PATH (active) STATE
1. TRAP DATA/COMMAND
2. TRANSLATE DATA/COMMAND
3. SEND DATA/COMMAND TO H.A.S.
4. RECEIVE DATA/COMMANDFROM H.A.S.
SAK PRESSED

C. TCBE OBJECT DIAGRAM



In the Object Diagram the lines indicate the associations, or the relationships between the software modules. The dashed lines are used to signify associations that will travel over physical communication links between hardware components of two modules. These links are only present in the interface portion between the TCBE and the host computer and in the secure local area network operations of the TCBE. Although the object diagram indicates that the TCBE Keyboard and Monitor are part of the TCBE, this is meant only as an abstraction. The associations do not represent dependencies between

modules, they serve only to represent the connections that either conceptually or physically link the components.

D. TCBE FUNCTIONAL MODULES DIAGRAM AND DEFINITIONS

The following table provides a brief description of each module's functions. It also provides a listing of the dependencies for each module.

PRELIMINARY TCBE DEPENDENCY AND ENGINEERING ARCHITECTURE DIAGRAM

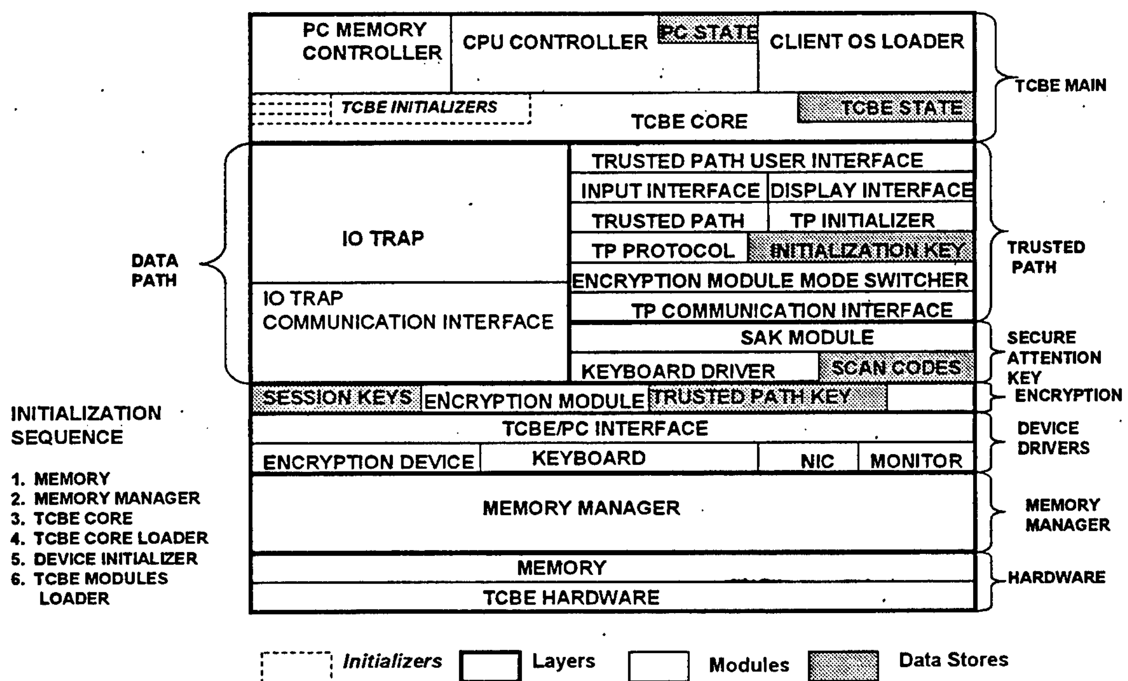


Table 1. Preliminary Module Definitions and Dependencies

Module Name	Module Function	Depends On
TCBE MAIN MODULES		
Host Computer/Client OS	Host computer that holds the client OS; contains a driver that sees the TCBE as just a NIC	The TCBE to function on the secure network
TCBE Core	Core of the TCBE; controls the initialization of the modules of the TCBE; maintains TCBE current state.	SAK Trusted Path Encryption Module Memory Manager Data Path
TCBE Modules Loader (TML)	Loads and initializes the TCBE modules	TCBE Core
Client OS Loader (COSL)	Loads the Client OS on the host Computer; may need to create a Ram drive file system	TCBE Core
TCBE Core Loader (TCL)	Loads the TCBE Core into the controller device's memory on startup	Memory Manager
PC Memory Controller (PCML)	Overwrites PC's memory	TCBE Core
CPU Controller (CPUC)	Places the PC's CPU into one of three modes 1. Idle (No instructions executing) 2. Active (Normal Operation of the CPU) 3. Stop (Shutdown the Computer for restart)	TCBE Core
Device Initializer	Initializes the TCBE's hardware	
TCBE MAIN DATABASES		
TCBE State	States that the TCBE is in TCBE States: 1. Initialization 2. Wait 3. Trusted Path 4. Data Path	TCBE Core Trusted Path Data Path
PC State	Present mode of host Computer CPU: 1. Idle 2. Active 3. Stop	CPU Controller

TRUSTED PATH MODULE		
Trusted Path User Interface (TPI)	Presents Trusted Path interface and command items to the user; Collects user info for the Trusted path	TCBE Keyboard and Monitor Input Interface Display Interface High assurance server

		TP Protocol
Input Interface	Driver to operate TCBE keyboard	SAK Module TCBE Keyboard
Display Interface	Driver to operate TCBE Monitor	TCBE Monitor
Trusted Path Initializer (TPINIT)	Module produces an initialization key and algorithm that starts the Trusted path protocol	TP Protocol
TP Confirmation Protocol	Confirms presence of the Trusted Path with the high assurance server	TP Protocol
TP Protocol	The protocol that establishes the Trusted Path between the TCBE and the TCB	Encryption Module
Encryption Module Mode Switcher (EMMS)	Switches the mode of the Encryption Module between Trusted Path Mode and the Appropriate session key.	TP Communication Interface
TP Communication Interface	Communications protocol between the Trusted Path Module and the Encryption Module to the NIC	TCBE Hardware
TRUSTED PATH MODULE DATABASES		
Initialization Key	Key produced by the TPINIT that initializes the Trusted Path	
User Information (User Info)	Memory Buffer where data is stored before encryption and transmission. Store of user identification and authentication information: 1. User Name 2. Password 3. Biometric Data 4. Desired Session Level 5. Present Session Level 6. Default Session Level	Depends on high assurance server.

SECURE ATTENTION KEY MODULE		
Secure Attention Key Module (SAK)	Module contains the routine necessary to recognize a keyboard combination as a SAK and send a start Trusted Path command to the TCBE core. This is coded as part of an interrupt routine in the TCBE Executive.	Keyboard Driver
Keyboard Driver	Low level keyboard routine that allows the SAK module to work; This code sits on top of normal keyboard driver and modifies the driver to provide the SAK code; May not be necessary depending on how the SAK module and the	TCBE Keyboard

	keyboard driver are implemented	
SECURE ATTENTION KEY MODULE DATABASES		
Scan Codes	The memory buffer where the scan codes are compared against the SAK	Keyboard Memory

DATA PATH MODULE		
Input Output Trap (IO Trap)	Translates user data into High Assurance Server Format; May convert client OS file system commands to High Assurance Server Commands; Directs User data to Encryption Module and NIC. Both the IO Trap and the IO Trap Communication Interface would be coded as part of the TCBE Driver within the client operating system.	IO Trap Communication Interface Encryption Module
IO Trap Communication Interface	Protocol to talk to Encryption Module	

ENCRYPTION MODULE		
Encryption Module	This module provides the interface between TCBE components, the physical Encryption device or program and the NIC. Encryption Module Mode 1. Trusted Path 2. Data Path 3. Initialization	Encryption Device(s) Memory Manager
ENCRYPTION MODULE DATABASES		
Trusted Path Key	Store for the trusted path key provided by the High Assurance Server.	Memory
Session Keys	Multiple storage for the session keys provided by the High Assurance Server	Memory

DEVICE DRIVERS		
TCBE/PC Interface	Virtual NIC to host computer interface	Memory Manager TCBE Hardware
Encryption Device	The physical device or software routine that performs the actual encryption of data. There may be multiple Encryption Devices	Memory Manager TCBE Hardware
Network Interface Card	The physical device driver that connects the TCBE to the secure LAN	Memory Manager TCBE Hardware
Keyboard	The physical device driver that	Memory Manager

	provides keyboard input to the TCBE	TCBE Hardware
Monitor	The physical device driver that allows the TCBE to display information on the screen	Memory Manager TCBE Hardware

MEMORY MANAGER	Module that manages the memory resources of the TCBE.	Memory

HARDWARE		
Memory	The physical memory provided for the TCBE for its operation	
TCBE hardware	The physical devices that make up the TCBE	

APPENDIX B. TCBE SOFTWARE SPECIFICATION DOCUMENT

The TCBE software specification document lists the TCBE's Modules and their interfaces. The specification document is partitioned by major TCBE module. Not all of the functions, particularly the Trusted Path and the host computer hardware control functions, have complete specifications. Those will need to be developed as part of future research which was detailed in Chapter VI.

1. TRUSTED COMPUTING BASE EXTENSION SOFTWARE DESCRIPTION

The Trusted Computing Base Extension (TCBE) is a platform designed to connect to a commercial off-the-shelf (COTS) personal computer or workstation. The TCBE will allow the personal computer to function as a client on a secure multilevel local area network. The TCBE will provide this functionality by meeting a subset of the Department of Defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC) standards for a Class B3 evaluated system [1].

2. TRUSTED COMPUTING BASE EXTENSION LAYERS

The TCBE's software modules are designed in a layered manner. The higher level modules depend on the lower level modules for operation. For example, the proper operation of the Trusted Path Module is dependent upon the Encryption Module performing its cryptographic operations. Chapter III, Section C. describes the layers of the TCBE from the most privileged layer to the least privileged.

3. TRUSTED COMPUTING BASE EXTENSION MODULES

Description: This is a software specification document. The descriptions presented here are a representation of the functionality of the software modules. They do not represent the complete architectural layering. The software programmer may find that due to hardware and software constraints that the internals of the system should change. Any modifications should allow the TCBE to complete the requirements described in Chapter II of this thesis.

Variables:

TCBE_State = {Initialization, Wait, Data_Path, Trusted_Path}
Record Data_Block_Type = [string Command, generic Data]
Record TCBE_Data_Block [string Command, Data_Block_Type Data_Block, integer Size of Data_Block]

3. 1 SAK MODULE

3.1.1 Description: The SAK Module provides the operation of the Secure Attention Key for the Trusted Computing Base Extension. It is dependent upon the keyboard driver within the Hardware Module for proper operation. This module will be coded as an interrupt within the TCBE's operating system.

3.1.2 Variables:

Boolean SAK_Pressed; //if SAK Button is pressed returns true

3.1.3 Functions:

3.1.3.1 SAK_Monitor_Keyboard
3.1.3.1.1 Inputs
3.1.3.1.1.1 Input 1: Scan Codes
3.1.3.1.2 Outputs
3.1.3.1.2.1 Output 1: SAK_Pressed
3.1.3.1.3 Return Value: Boolean

3.1.3.1.4 Description: Check scan codes for SAK keys. If the correct buttons are pressed then send true. Continuous monitoring until the SAK keys are pressed. This function should be coded as an interrupt within the TCBE's operating system.

3.2 TRUSTED PATH MODULE

3.2.1 Description: The Trusted Path Module contains the functions and protocols necessary to operate the Trusted Path between the TCBE and the high assurance server. It depends on the Encryption Module, the SAK Module, and the Hardware Module for proper operation. The functions within this module are vague and represent an abstraction as to what will physically happen at the TCBE. They may change considerably depending on how the Trusted Path is actually constructed.

3.2.2 Variables:

Trusted_Path_Status = {Established, Not_Established}
Trusted_Path_State = {Idle, Confirming, Initializing, Logging_Out, Logging_In, Switching_Session_Level, Clearing_Buffer)
Trusted_Path_Command = {Start_Session, Reattach_Session, Login, Logout, Switch_Session_Level, Confirm, TP_Idle}
Encryption_Key Session_Encryption_Key
Integer Session_Number

Record Trusted_Path_Session_Info = [Session_Number, Session_Encryption_Key]
Record Trusted_Path_Session_Table = [Current_Session_Number, Trusted_Path_Session_Info]

Record Trusted_Path_Buffer =[integer Desired_Session_Level]
Record Current_User_Information = [Username, Password, Biometric_Data

3.2.3 Functions:

3.2.3.1 TP_Start_Trusted_Path
3.2.3.1.1 Inputs
3.2.3.1.1.1 Input 1: TCBE_State
3.2.3.1.2 Outputs
3.2.3.1.2.1 Output 1: None
3.2.3.1.3 Return Value: Boolean
3.2.3.1.4 Description: This function initiates the Trusted Path. It checks to ensure that the TCBE and the host computer are in the correct state in order to begin.

3.2.3.2 Idle_Trusted_Path
3.2.3.2.1 Inputs
3.2.3.2.1.1 Input 1: None

3.2.3.2.2 Outputs
3.2.3.2.2.1 Output 1:
3.2.3.2.3 Return Value: Trusted_Path_Command
3.2.3.2.4 Description: This function displays a Trusted Path Interface screen on the monitor.

Pseudo-code

```
Trusted_Path_State = Idle;
Loop {
  TPI_Display_TP_Menu;
  If Command = Confirm
    Then TP_Confirm_Trusted_Path;
  If Command = Login
   Then TP_Login;
  If Command = Switch_Session_Level;
   Then TP_Switch_Session_Level;
  Exit when Command = Start_Session, Reattach_Session, or Logout
}
return (Command);
```

3.2.3.3 TP_Confirm_Trusted_Path
3.2.3.3.1 Inputs
3.2.3.3.1.1 Input 1: None
3.2.3.3.2 Outputs
3.2.3.3.2.1 Output 1: None
3.2.3.3.3 Return Value: Boolean
3.2.2.3.4 Description: This function confirms that a trusted path exists to the user.

Pseudo-code

```
TP_Clear_Buffers;
Trusted_Path_State = Confirming;
"Code to confirm trusted path";

If confirmation successful then return to calling function
  Else Trusted_Path_Status = Not_Established;
TP_Start_Trusted_Path;
```

3.2.3.4 TP_Login
3.2.3.4.1 Inputs
3.2.3.4.1.1 Input 1: None
3.2.3.4.2 Outputs

3.2.3.4.2.1 Output 1: Username
3.2.3.4.2.2 Output 2: Password and/or Biometric data
3.2.3.4.2.3 Output 3: session level
3.2.3.4.3 Return Value: Boolean
3.2.3.4.4 Description: This function handles the logging in to the high assurance server. This function does not perform the actual login. It only forwards the encrypted information to the high assurance server.

3.2.3.5 TP_Logout
3.2.3.5.1 Inputs
3.2.3.5.1.1 Input 1: None
3.2.3.5.2 Outputs
3.2.3.5.2.1 Output 1: None
3.2.3.5.3 Return Value
3.2.3.5.4 Description: This function handles the logout of the high assurance server. It does not perform the actual log out, only forwards the command and then waits for the receipt of the confirmed logout signal from the high assurance server. The function will ensure that the all the buffers within the Trusted Path Module are cleared.

3.2.3.6 TP_Switch_Session_Level
3.2.3.6.1 Inputs
3.2.3.6.1.1 Input 1:integer New_Session_Number
3.2.3.6.2 Outputs
3.2.3.6.2.1 Output 1:
3.2.3.6.3 Return Value: Boolean
3.2.3.6.4 Description: This function performs the switching of the session level at the TCBE. It will get the appropriate session key, then make the call to start a new session.

Pseudo-code

```
TP_Get_Session_Key;
Trusted_Path_Session_Table.Current_Session_Number =
        New_Session_Number;
TP_Start_Session;
```

3.2.3.7 TP_Start_Session
3.2.3.7.1 Inputs
3.2.3.7.1.1 Input 1: integer Session_Level
3.2.3.7.2 Outputs

3.2.3.7.2.1 Output 1: None
3.2.3.7.3 Return Value: Boolean
3.2.3.7.4 Description: This function makes sure that the TCBE is ready to start a session. It will load the appropriate encryption key into the Encryption Module and then start the Data Path.

Pseudo-code

```
If (EM_Mode_Switch(Data_Path,
        Trusted_Path_Session_Table.Current_Session_Number))
  Then TCBE_Switch_State(Data_Path);
        TCBE_State = Data_Path;
Else Return (Encryption module error)
```

3.2.3.8 TP_Get_Session_Key
3.2.3.8.1 Inputs
3.2.3.8.1.1 Input 1: None
3.2.3.8.2 Outputs
3.2.3.8.2.1 Output 1: Session_Encryption_Key
3.2.3.8.3 Return Value:
3.2.3.8.4 Description: This function gets the session encryption key from the high assurance server.

3.2.3.9 TP_Clear_Buffers
3.2.3.9.1 Inputs
3.2.3.9.1.1 Input 1: None
3.2.3.9.2 Outputs
3.2.3.9.2.1 Output 1: None
3.2.3.9.3 Return Value:
3.2.3.9.4 Description: This function clears all memory buffers used by the Trusted Path Modules.

3.2.3.10 TPI_Display_TP_Menu
3.2.3.10.1 Inputs
3.2.3.10.1.1 Input 1: None
3.2.3.10.2 Outputs
3.2.3.10.2.1 Output 1: None
3.2.3.10.3 Return Value:
3.2.3.10.4 Description: This function displays a Trusted Path Interface Menu on the TCBE's monitor. It also provides the mechanism by which user

information is entered from the keyboard and sent to the high assurance server and the Trusted Path module.

3.3 TCBE_CORE MODULE

3.3.1 Description: The TCBE Core module provides the functions necessary to interface with the host computer. It is dependent on all the other modules of the TCBE for correct operation. The descriptions here are intended to provide a sketch of the TCBE Core's functionality. They do not represent a complete architectural layering.

3.3.2 Variables:

Integer Session_Number //One assigned to each session for the current user
Record Client_OS_Information // particular state of the CPU and OS when halted by TCBE. Note: to be used when TCBE has ability to hold Client OS state.
Record Session_Info [Session_Number, Client_OS_Information]
Record Session_Table [integer Current_Session_Number, Array of Session_Info]
Data_Path_Mode = {Initial, Active}
Client_OS_Status = {Loaded, Not_Loaded}
CPU_Command = {Activate, Idle, Stop}
Data_Path_Command = {Start, Stop}
PC_State = {Idle, Active, Stop}
Module = {Data_Path, Encryption_Module, TP_Module, SAK_Module}

3.3.3 Functions:

3.3.3.1 TCBE_Switch_State
3.3.3.1.1 Inputs
3.3.3.1.1.1 Input 1: TCBE_State
3.3.3.1.2 Outputs
3.3.3.1.2.1 Output 1: None
3.3.3.1.3 Return Value: Boolean Result
3.3.3.1.4 Description: This function changes the TCBE State and its functioning to the called upon state. Once this operation is complete it will return true if the switch was successful.

Pseudo-code

```
Result = False;
If TCBE_State = Data_Path
  Then Result = TCBE_CPU_Controller(Activate, 0);
If Client_OS_Status = Not_Loaded
  Then if Result = (TCBE_Initialize_Data_Path)
        Then Result = DP_Path(Open);
If Client_C  _Status = Loaded
```

```
    Then Result = DP_Path(Open);

If TCBE_State = Trusted_Path
  Then Result = DP_Path(Close);
       Result = TCBE_CPU_Controller(Idle, 0);

If TCBE_State = Wait
  Then Result = DP_Path(Close);
              Result = TCBE_CPU_Controller(Stop, 0);

If TCBE_State = Initialization
  Then Result = TCBE_Initialize;
Return(Result)
```

3.3.3.2 TCBE_Initialize_Data_Path
3.3.3.2.1 Inputs
3.3.3.2.1.1 Input 1: None
3.3.3.2.2 Outputs
3.3.3.2.2.1 Output 1: None
3.3.3.2.3 Return Value: Boolean
3.3.3.2.4 Description: This function activates the host computer and ensures that the client operating system is loaded.

Pseudo-Code

```
If (TCBE_CPU_Controller(Activate,0));
If (TCBE_Load_Client_OS(0))
  Then Client_OS_Status = Established;
```

3.3.3.3 TCBE_Activate_Data_Path
3.3.3.3.1 Inputs
3.3.3.3.1.1 Input 1: integer Current session number
3.3.3.3.2 Outputs
3.3.3.3.2.1 Output 1:
3.3.3.3.3 Return Value: Boolean
3.3.3.3.4 Description: This function will load the correct operating system state depending on the session level. This function is used when switching session levels. (For future work when capable of saving Client OS state For each session level)

3.3.3.4 TCBE_Load_Client_OS
3.3.3.4.1 Inputs
3.3.3.4.1.1 Input 1: integer Session Number
3.3.3.4.2 Outputs
3.3.3.4.2.1 Output 1: None

3.3.3.4.3 Return Value: Boolean
3.3.3.4.4 Description: This function loads the Client OS into the host computer

3.3.3.5 TCBE_Module_Loader
3.3.3.5.1 Inputs
3.3.3.5.1.1 Input 1: None
3.3.3.5.2 Outputs
3.3.3.5.2.1 Output 1:
3.3.3.5.3 Return Value: Boolean
3.3.3.5.4 Description: This module loads each module of the TCBE in a distinct memory location within the controller card. This will be important if the entire TCBE must be coded in assembly.

3.3.3.6 TCBE_CPU_Controller
3.3.3.6.1 Inputs
3.3.3.6.1.1 Input 1: Command
3.3.3.6.1.2 Input 2: integer Session_Number
3.3.3.6.2 Outputs
3.3.3.6.2.1 Output 1: None
3.3.3.6.3 Return Value: Boolean
3.3.3.6.4 Description: This function controls the host computer's processor and memory. May be coded as an interrupt to the host computer.

Pseudo-Code:

```
If Command = Idle
  Then TCBE_Save_PC_State(Session_Number); //if possible
       TCBE_Erase_Secondary_Memory;

If Command = Activate
  Then TCBE_Start_PC_State(Session_Number);

If Command = Stop
  Then TCBE_Erase_Secondary_Memory;

Return(Boolean);
```

3.3.3.7 TCBE_Initialize
3.3.3.7.1 Inputs
3.3.3.7.1.1 Input 1: None
3.3.3.7.2 Outputs
3.3.3.7.2.1 Output 1: None
3.3.3.7.3 Return Value: Boolean

3.3.3.7.4 Description: This function provides the initialization routines for the TCBE.

3.3.3.8 TCBE_Save_PC_State
3.3.3.8.1 Inputs
3.3.3.8.1.1 Input 1: integer Session Number
3.3.3.8.2 Outputs
3.3.3.8.2.1 Output 1: None
3.3.3.8.3 Return Value: Boolean
3.3.3.8.4 Description: This function saves the state of the PC and registers when the Trusted Path is invoked. This function may not be necessary if the TCBE_CPU_Controller is constructed as an interrupt. However, if future work were to allow multiple sessions without rebooting between sessions then this function would have to be developed.

3.3.3.9 TCBE_Start_PC_State
3.3.3.9.1 Inputs
3.3.3.9.1.1 Input 1: integer Session Number
3.3.3.9.2 Outputs
3.3.3.9.2.1 Output 1: None
3.3.3.9.3 Return Value: Boolean
3.3.3.9.4 Description: This module loads the state of the PC according to the Session Number. When multiple sessions are not possible, the Session Number will always be zero indicating a reload of the entire system.

3.3.3.10 TCBE_Erase_Secondary_Memory
3.3.3.10.1 Inputs
3.3.3.10.1.1 Input 1: None
3.3.3.10.2 Outputs
3.3.3.10.2.1 Output 1: None
3.3.3.10.3 Return Value: Boolean
3.3.3.10.4 Description: This function erases the host computer memory by writing a random pattern of one's and zeroes in the host computer's RAM and caches.

3.3.3.11 Function: TCBE_Display
3.3.3.11.1 Inputs
3.3.3.11.1.1 Input 1: None
3.3.3.11.2 Outputs
3.3.3.11.2.1 Output 1: None
3.3.3.11.3 Return Value: None

3.3.3.11.4 Description: This function displays a blank screen before the Trusted Path has been started.

3.4 ENCRYPTION MODULE

3.4.1 Description: The Encryption Module provides those functions that interface with the encryption devices or algorithms used by the TCBE.

3.4.2 Databases: EM_Data_Buffer = Array of TCBE_Data_Block

3.4.3 Functions:

3.4.3.1 EM_Mode_Switch
3.4.3.1.1. Inputs
3.4.3.1.1.1 Input 1: TCBE_State
3.4.3.1.1.2 Input 2: Session_Number
3.4.3.1.2 Outputs
3.4.3.1.2.1 Output 1: None
3.4.3.1.3 Return Value: Boolean
3.4.3.1.4 Description: Switches the encryption module to the sent in state and to the corresponding session key if applicable. Returns true once this operation is complete

3.4.3.2 EM_Send
3.4.3.2.1 Inputs
3.4.3.2.1.1 Input 1: TCBE_Data_Block Black_Data
3.4.3.2.1.2 Input 2: Integer EM_Socket
3.4.3.2.2 Outputs
3.4.3.2.2.1 Output 1: None
3.4.3.2.3 Return Value: integer
3.4.3.2.4 Description: This function takes the data and encrypts it before sending it to the Communications device. It then takes the return from the communications device and decrypts it. The decrypted data is assigned a buffer position corresponding to the EM_Socket.

Pseudo-code

```
TCBE_Data_Block Red_Data

CD_Encrypt (TCBE_Data_Block, Black_Data, key);
NIC_Write (Comm_Port_No, Black_Data, size)
NIC_Read (Comm_Port_No, Buffer, size, Black_Data);
CD_Decrypt (Black_Data, Red_Data, key);
EM_Assign_Data (Red_Data, EM_Socket);
Return(integer);
```

3.4.3.3 EM_Receive
3.4.3.3.1 Inputs

3.4.3.3.1.1 Input 1: Integer EM_Socket
3.4.3.3.2 Outputs
3.4.3.3.2.1 Output 1: TCBE_Data_Block
3.4.3.3.3 Return Value: integer
3.4.3.3.4 Description: This function returns the size of the data located in the EM_Data_Buffer at the EM_Socket. The data has been copied to the TCBE_Data_Block

3.4.3.4 EM_Assign_Data
3.4.3.4.1 Inputs
3.4.3.4.1.1 Input 1: TCBE_Data_Block Red_Data
3.4.3.4.1.2 Input 2: Integer EM_Socket
3.4.3.4.2 Outputs
3.4.3.4.2.1 Output 1: None
3.4.3.4.3 Return Value: None
3.4.3.4.4 Description: This function assigns the data to a spot on the EM_Data_Buffer

Pseudo-code

```
EM_Data_Buffer(EM_Socket) = Red_Data
```

3.5 DATA PATH MODULE

3.5.1 Description: This module provides the functions to operate the Data_Path. It depends on the Encryption Module and the Hardware Module to operate correctly. This module would serve as an interface to a TCBE Driver that is installed in the client operating system.

3.5.2 Databases: None

3.5.3 Functions:

3.5.3.1 DP_Path
3.5.3.1.1 Inputs
3.5.3.1.1.1 Input 1: Command{Open, Close}
3.5.3.1.2 Outputs
3.5.3.1.2.1 Output: Boolean
3.5.3.1.3 Description: This function starts the Data Path between the host computer and the high assurance server. The data path is merely a pipe with one end the Client OS and the other end the Encryption Module. When this pipe is opened the Client OS receives a signal the path is ready. When the pipe is closed the Client OS can no longer send data to the High Assurance Server

3.5.3.2 DP_Get
3.5.3.2.1 Inputs
3.5.3.2.1.1 Input 1: string Filename
3.5.3.2.2 Outputs
3.5.3.2.2.1 Output 1: None
3.5.3.2.3 Return Value: FILE Pointer
3.5.3.2.4 Description: This function gets a block of data or file from the high assurance server via the TCBE and a data transfer protocol. It uses the Trusted Computing Base Data Transfer Protocols (tcbdtp) discussed in Chapter III.

Pseudo-code

```
Convert tcbdtp Get function into TCBE_Data_Block
EM_Send(TCBE_Data_Block, EM_Socket)
EM_Receive(TCBE_Data_Block, EM_Socket)
Pointer = Convert TCBE_Data_Block into file

Return(Pointer)
```

3.5.3.3 DP_Send
3.5.3.3.1 Inputs

3.5.3.3.1.1 Input 1: Data
3.5.3.3.2 Outputs
3.5.3.3.2.1 Output 1: None
3.5.3.3.3 Return Value: None
3.5.3.3.4 Description: This function gets a block of data or file from the host computer and sends it to the high assurance server via the encryption module. It uses the Trusted Computing Base Data Transfer Protocols (tcbdtp) discussed in Chapter III. Whether this function waits for an acknowledgement is up to the designers of the tcbdtp or other communication protocols.

Pseudo-code

```
Convert tcbdtp Send function into TCBE_Data_Block
EM_Send(TCBE_Data_Block, EM_Socket)
```

3.6 TCBE HARDWARE

3.6.1 Description: The TCBE Hardware Module provides the interfaces between TCBE modules and the hardware of the Trusted Computing Base Extension.

3.6.2 Databases:

3.6.3 Functions

3.6.3.1 CD_Encrypt
3.6.3.1.1 Inputs
3.6.3.1.1.1 Input 1: TCBE_Data_Block Red_Data
3.6.3.1.1.2 Input 2: Encryption_Key key
3.6.3.1.2 Outputs
3.6.3.1.2.1 Output 1: TCBE_Data_Block Black_Data
3.6.3.1.3 Return Value: Boolean
3.6.3.1.4 Description: This function sends the data to the cryptographic device for encryption.

3.6.3.2 Function: CD_Decrypt
3.6.3.2.1 Inputs
3.6.3.2.1.1 Input 1: TCBE_Data_Block Black_Data
3.6.3.2.1.2 Input 2: Encryption_Key key
3.6.3.2.2 Outputs
3.6.3.2.2.1 Output 1: TCBE_Data_Block Red_Data
3.6.3.2.3 Return Value: Boolean
3.6.3.2.4 Description: This function sends the data to the cryptographic device for decryption.

3.6.3.3 Function: NIC_Write
3.6.3.3.1 Inputs
3.6.3.3.1.1 Input 1: Integer Comm_Port_No
3.6.3.3.1.2 Input 2: TCBE_Data_Block Black_Data
3.6.3.3.1.3 Input 3: Integer size of data
3.6.3.3.2 Outputs
3.6.3.3.2.1 Output 1: None
3.6.3.3.3 Return Value: Boolean
3.6.3.3.4 Description: This function sends encrypted data to the communications device to send on to the net

3.6.3.4 Function: NIC_Read
3.6.3.4.1 Inputs
3.6.3.4.1.1 Input 1: Integer Comm_Port_No
3.6.3.4.1.2 Input 2: TCBE_Data_Block Buffer
3.6.3.3.1.3 Input 3: Integer size of data
3.6.3.4.2 Outputs
3.6.3.3.2.1 Output 1: TCBE_Data_Block Black_Data
3.6.3.3.3 Return Value: Boolean
3.6.3.3.4 Description: This function receives encrypted data from the communications device.

APPENDIX C. TRUSTED COMPUTING BASE EXTENSION EXECUTIVE SOURCE CODE

The source code contains the files necessary to load and operate the implemented subset of the Trusted Computing Base Extension. The files provide include the C language headers and source code as well as the assembly language modules. The makefile for the project is included as well as a batch file. The batch file is used to overcome the conflict between long file names and the eight character file name limitations placed on compiling and linking the code on Microsoft Disk Operating System (MSDOS).

1. IMPLEMENTATION FUTURE WORK

Most of the functions that are identified in the specification were not built for this implementation and are dependent on the future work mentioned in the Conclusions chapter. However, there are several basic operating functions that are missing from this implementation that are needed to produce a fully operational Trusted Computing Base Extension.

a. Interrupt Service Routines

Interrupt Service Routines (ISR) are needed to manipulate the TCBE's keyboard. These ISR's are hardware dependent. Currently, the Secure Attention Key is tested by using'a test function: SAKTest, which causes the event to happen within the TCBE's executive.

b. Memory Manager

The memory functions present in this implementation are simple and static. Starting at a static starting address, a constant size list of fixed-size memory blocks is built. Each state is assigned a memory block in sequential order, i.e. state 1 gets memory block 1, state 2 gets memory block 2, etc. When the state is finished with the block, the block is designated as free. However, the block is never used again by another state. If a state needs a new block of memory, it uses the next free sequential block of memory on the memory list. For example if there are five states, and state 2 frees its block and then needs another block of memory it will be assigned memory block 6. This happens until all of the memory blocks have been used once. Then the system is out of memory and no longer operates. A dynamic memory management system needs to be developed. The dynamic memory manager should be able to use blocks of different size, and allow states to reuse different blocks of memory after the memory has been overwritten by an erasing function.

2. ASSEMBLY LANGUAGE TO C LANGUAGE LINKAGE

The code developed for the TCBE was written in the C and Assembly programming languages. Because of this linkage several, there are several requirements that must be fulfilled before modifying, compiling, and linking the source code.

a. Resolving Memory and Linkage Issues

In order for the names of functions that are called across different language models to be recognized, the following switches have to be applied when assembling and compiling the source code.

Assembly code: TASM /mx code.asm

TASM is the name of the assembler. In this case Borland's Turbo Assembler. The /mx switch that tells the assembler not to change the case of the names of the procedures that are present in the Assembler code. TASM will automatically change all characters to uppercase. The C language does not change the case of its functions. If this switch is not used, the linking operation will not work.

C Code: BCC –c –ml code.c

BCC is the name of the compiler. In this instance it is the Borland C compiler. The –c switch tells the compiler not to link the code, this will be accomplished in the batch and make files. The –ml switch tells the compiler to compile the code using the LARGE model. The LARGE model means that the program's code and data reside in different segments in memory. This is necessary because of the size of the code and the fact that it is written in two different languages. The assembler code must have the ".model large" or similar statement present at the beginning of its program.

b. Resolving Shared Procedure and Function Names

In addition to not changing the case of its functions when the code is compiled, the C language adds underscores to the function names in the object code file that is produced after compilation. Since Assembly language does not add the underscore to its procedure names, the underscore must be added to all procedures that are called from C language modules and the underscore must be added to Assembly procedures that are called by C modules. For example:

Assembly Program

```
public _function1   ; This is an Assembly function called by a C module

extrn _function2    ; This is a C function called by an Assembly module.

<beginning of Assembly Program Body>

_function1 proc far
        <body of _function1 >
_function1 endp


<ending of Assembly Program Body>
```

In the C language module, _function2 will be written as function2.

3. TCBE IMPLEMENTATION SOURCE CODE

a. Custom.h

```
/* Filename: Custom.h
   Author: Micheal Podanoffsky; modified by Jason Hackerson
   Date: 04 August 1998
   Function: Header file for customizable options of the executive
*/


/*----------------------------------------------------------------
   Customizable Parameters
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Make sure all parameters here are customized to your needs.
   ----------------------------------------------------------------*/


#define NUM_STATES          12  /* -- number of states ------ */

#define MEM_BLOCKS              (NUM_STATES * (NUM_STATES/2))

#define NUM_EVENTS          7   /* -- number of events ------- */

#define MAX_STATE_NAME      32  /* -- max characters in state name -- */


#define MIN_STACK_SIZE      128

/* Serial Communications Events - Not used in this implementation of the
   TCBEX */

                          /* -- enter in the enum list the
                             names of any custom event flags -- */
typedef enum {

   NOT_WAITING = 0,          /* keep this line here */
   COM1_RING,
   COM1_TX_EMPTY,
   COM1_DATA_AVAILABLE,
   COM1_CHANGE_CARRIER,
```

```
  COM2_RING,
  COM2_TX_EMPTY,
  COM2_DATA_AVAILABLE,
  COM2_CHANGE_CARRIER,

  COM3_RING,
  COM3_TX_EMPTY,
  COM3_DATA_AVAILABLE,
  COM3_CHANGE_CARRIER,

  COM4_RING,
  COM4_TX_EMPTY,
  COM4_DATA_AVAILABLE,
  COM4_CHANGE_CARRIER,

  LAST_EVENT              /* -- keep this entry alone -- */
  } customEvents;
```

b. TCBEX.h

```
/* Filename: tcbex.h
  Author: Micheal Podanoffsky; Modified by Jason Hackerson
  Date: 04 August 1998
  Function: Provides Header file for tcbe executive
*/
#include "Custom.h"


/* //////////// System Events //////////////////////////////////// */
/*--- These are events that serve as inputs into the state machine engine ---*/

typedef enum {

  SAK_EVENT,
  TRANSIT_EVENT,
  REATTACH_EVENT,
  KEYBOARD_WAIT,
  LOGIN_EVENT,
  LOGOUT_EVENT,
  SWITCH_SESSION_EVENT,
  LAST_SYSTEM_EVENT

  } systemEvents;
```

```
/* TCBEX system states */
typedef enum {

  TCBE_INIT,
  TCBE_DISPLAY,
  TRUSTED_PATH,
  TP_INIT,
  TP_CONFIRM,
  TP_DISPLAY,
  TP_LOGIN,
  TP_LOGOUT,
  TP_SWITCH,
  DP_INIT,
  DP_ACTIVE

}stateTypes;

typedef enum {

    REATTACH,
    CONFIRM,
    LOGIN,
    LOGOUT,
    SWITCH_SESSION

}commandTypes;
```

```
/*--- TCBEX system flags system flags help the state machine engine keep track
 of what has been initialized ---*/
typedef enum {

  TCBE_INIT_FLAG,
  TP_INIT_FLAG,
  DP_INIT_FLAG

}flagTypes;


/* //////////// Other Defines ///////////////////////////////////// */

#define FALSE           0
#define TRUE            1

#define NULL                0L

#define PERIODIC            1

#define CURR_STATE              0L          /* indicates current task */
#define CURRENT_STATE              CURR_STATE     /* indicates current task */
#define CURRENTSTATE               CURR_STATE     /* indicates current task */

#define NO_ARGUMENT              NULL

#define event               unsigned int   /* event flags stored in unsigned ints */
#define far_address(s, p)     (void *)((unsigned long)((unsigned long)((s)) << 16) +
(unsigned long)((p)))

extern unsigned long *SystemClock;

/* ///////////////////////////////////////////////////////////////////
  Error Codes

/////////////////////////////////////////////////////////////////// */

#define SUCCESS               0
#define ERR_TASKALLOC_FULL       SUCCESS - 1
#define ERR_TIMERID_NOTFOUND      SUCCESS - 2
#define ERR_INVALID_ADDRESS      SUCCESS - 3
#define ERR_INVALID_EVENTID      SUCCESS - 4
```

```
/* //////////////////////////////////////////////////////////////////
  State Structure
  ----------------------------------

  This control structure defines a State.  A state is a grouping of functions,
  that determines the TCBE's current Mode of operations.  When a state is
  suspended, all of the registers including instruction address, flags, etc...
  are stored in the state's stack and the suspended bit is set.  The
  suspended bit indicates that the stack contains registers to pop
  before starting the state.  The timing variables are left in the structure to
  support a future version of the TCBE that may use multitasking.

////////////////////////////////////////////////////////////////// */

typedef struct {
    unsigned          suspended:1;
    unsigned          free:1;

    unsigned long     sleep;
    unsigned long     time_slice;
    unsigned long     time_slice_interval;
    event             events[ NUM_EVENTS ];
    event             startup_event;

    void              *init_stack_addr;
    void              *curr_stack_addr;
    void              *start_addr;
    void              *argument;
    unsigned          stack_size;
    int               state_num;
    char              state_name [ MAX_STATE_NAME ];

    }State;


/* //////////////////////////////////////////////////////////////////
  Function Prototypes

////////////////////////////////////////////////////////////////// */

/* -- general prototypes ------------------------------------------- */

void  initStateSystem(void );
void  restore_IntTraps(void );
void  * tmalloc(void );
```

```
void     tstrncpy(char * str1, char * str2, int n);
void   * tfree(void * ptr);


extern void far *  save_initregs
                  ( void  * stack,
                    void  * start_addr,
                    void  * argument,
                    int state_num );
extern void  tprint(char * string);
extern void  init_IntTraps(void );
extern void  setClockInterrupt( unsigned long min_clock_value );
extern void  setClockIntValue(void );


extern void  scheduler(void );
extern void  enableInts(void );
extern void  condenableInts(int );
extern int   disableInts(void );

extern void  enableSched(void );
extern void  disableSched(void );

extern event  SystemEvent[];

extern unsigned long *SystemClock;

/* -- event related prototypes ------------------------------------ */

void   clearSystemEvent( event event_id );
int   defineEventFct( event event_id,
                      int ( *start_addr) (event event_id, void * arg ),
                      void * argument );
void Transit(void );
void SAKTest(void );
void Reattach(void );
void Login(void );
void Logout(void );
void Switch_Session(void );
```

```
/* -- State related prototypes --------------------------------------- */

State * defineState(int ( *start_addr) (void * arg ),
                    void * argument,
                    unsigned stack_size,
                    int state_num,
                    char * state_name );

State * redefineState(State * state);

void transit(int current_state, int new_state);
void *  returnState_Stackframe(void );
int   terminateState(State * state );
int   terminateState_num(int state_num );

typedef struct {
   unsigned int ax;
   unsigned int bx;
   unsigned int cx;
   unsigned int dx;
   unsigned int si;
   unsigned int di;
   unsigned int es;
   unsigned int ds;
   } ALL_REGS;

void  SaveRegisters(void );
void  RestoreRegisters(void );
```

c. TCBEX.c

```
/* ////////////////////////////////////////////////////////////////////

Name: TCBEX.c (Trusted Computing Base Extension Executive)
Author: Jason Hackerson (from code originally written
                by  Michael Podanoffsky)
Date: 10 September 1998
Function: Provides the operating executive that operates the Trusted Computing
        Base Extension.Note: Some functions remain from original RTX code. These
functions are deemed potentially useful for future implementations of the TCBEX.

//////////////////////////////////////////////////////////////////// */
```

```
/*////////////////////////////////////////////////////////////////////

    Include these libraries when working with DOS and need their
  functionality. Not needed otherwise

#include <stdio.h>
#include <malloc.h>

////////////////////////////////////////////////////////////////////*/
#include "tcbex.h"

/* ////////////////////////////////////////////////////////////////////
  tcbex internal data structures

//////////////////////////////////////////////////////////////////// */

int Current_State_Array[ NUM_STATES ];
int Event_Array[ NUM_EVENTS ];
State * State_Matrix[ NUM_EVENTS ][NUM_STATES];
stateTypes TCBE_state;
systemEvents TCBE_event;
flagTypes TCBE_flag;
int System_flags[3];
State  * currentState;
State States[NUM_STATES];

/*////////////////////////////////////////////////////////////////////

  Left over from RTX. May be useful to define events that are more
  robust than current events

////////////////////////////////////////////////////////////////////*/
typedef struct {
    int (*  fct)(event event_id, void *arg);
    void *argument ;

    } eventFct;

/* ////////////////////////////////////////////////////////////////////
   Allocate States and Counters

//////////////////////////////////////////////////////////////////// */
int actual_states = 0;
State *currentState = NULL;
unsigned long  *SystemClock = far_address(0x0040, 0x006C);
```

```
/*////////////////////////////////////////////////////////////////////
 Data structures for memory management

////////////////////////////////////////////////////////////////////*/

struct Block_Type{

 int block_num;
 void * block_address;
 int free;

};

struct Block_Type Block_Array[20];

#define NOT(a)                  ~(a)
#define PROTECTION_PATTERN      0xC0C4          /* stack protection signature */


/* ////////////////////////////////////////////////////////////////////

  defines an event function.
  Us a null fct argument to clear an event function.
  Not used by TCBEX; may be useful for future implementations

////////////////////////////////////////////////////////////////////
int defineEventFct( event event_id,
            int ( *fct) (event event_id, void  * arg ),
            void  * argument )
{

  if ( event_id > NUM_EVENTS )
    return ERR_INVALID_EVENTID;

  EventFunctions[ event_id ].fct = fct;
  EventFunctions[ event_id ].argument = argument;
  return SUCCESS;
}
*/
```

```
/* ///////////////////////////////////////////////////////////////////////

  Sets any system event

/////////////////////////////////////////////////////////////////////// */
void  setSystemEvent( unsigned event_id )
{

  disableInts();
  Event_Array[event_id] = 1;
  enableInts();
}


/* ///////////////////////////////////////////////////////////////////////

  waits for keyboard event

/////////////////////////////////////////////////////////////////////// */
extern void waitKbdEvent(void )
{

  setSystemEvent( KEYBOARD_WAIT );
}

/* ///////////////////////////////////////////////////////////////////////

  Sets keyboard event in SystemEvent

/////////////////////////////////////////////////////////////////////// */
extern void   setKbdEvent(void )
{

  setSystemEvent( KEYBOARD_WAIT );
}
/* ///////////////////////////////////////////////////////////////////////

  Sets SAK event in SystemEvent

/////////////////////////////////////////////////////////////////////// */
extern void setSAKEvent(void )
{

  setSystemEvent( SAK_EVENT);
```

```
}

void SAKTest(void )
{

    setSAKEvent();
    scheduler();
}
/* ////////////////////////////////////////////////////////////////////

  Sets Transit event in SystemEvent

//////////////////////////////////////////////////////////////////// */
extern void setTransitEvent(void )
{

  setSystemEvent( TRANSIT_EVENT );

}

void Transit(void ){

  setTransitEvent();
  scheduler();
}

/* ////////////////////////////////////////////////////////////////////

  Sets Login event in SystemEvent

//////////////////////////////////////////////////////////////////// */
extern void setLoginEvent(void )
{

  setSystemEvent( LOGIN_EVENT );

}

void Login(void ){

  setLoginEvent();
  scheduler();
}
/* ////////////////////////////////////////////////////////////////////
```

```
  Sets Logout event in SystemEvent

////////////////////////////////////////////////////////////////// */
extern void setLogoutEvent(void )
{

  setSystemEvent( LOGOUT_EVENT );

}

void Logout(void ){

  setLogoutEvent();
  scheduler();
}
/* //////////////////////////////////////////////////////////////////

  Sets Switch Session event in SystemEvent

////////////////////////////////////////////////////////////////// */
extern void setSwitchSessionEvent(void )
{

  setSystemEvent( SWITCH_SESSION_EVENT );

}

void Switch_Session(void ){

  setSwitchSessionEvent();
  scheduler();
}

/* //////////////////////////////////////////////////////////////////

  Sets Reattach event in SystemEvent

////////////////////////////////////////////////////////////////// */
extern void setReattachEvent(void )
{

  setSystemEvent( REATTACH_EVENT );

}
```

```
void Reattach(void ){

  setReattachEvent();
  scheduler();
}


/* ////////////////////////////////////////////////////////////////////

  Support routine: returns hello as a place marker when debugging
  assembly code

//////////////////////////////////////////////////////////////////// */
void hello(void ){

tprint(" Hello ");

}

/*////////////////////////////////////////////////////////////////////

        Function returns the stack pointer for the current stack when
    switching stacks.
    printf statement requires stdio.h

////////////////////////////////////////////////////////////////////*/
extern void *  returnState_StackFrame(void )
{

  /*printf(" \nRSSF  %#08.4lx \n", (int *)currentState->curr_stack_addr);*/
  return ( currentState ? currentState->curr_stack_addr : NULL);
}

/* ////////////////////////////////////////////////////////////////////

  Support routine: sets stack address of a suspended stack.
        printf statement requires stdio.h

//////////////////////////////////////////////////////////////////// */
void  saveCurrentState_Stack(void  * stack)
{

  if ( currentState && stack )
    currentState->curr_stack_addr = stack;
```

```
  /*printf("SCCS stack addr %#08.4lx \n",(int *)currentState->curr_stack_addr);*/

}

/*////////////////////////////////////////////////////////////////////////

      Debugging function that provides an address. Needs stdio.h to display
    the address.

////////////////////////////////////////////////////////////////////////*/
void  printa(void * address)
{
  tprint(" Printing Address ");
  /*printf("address =  %#08.4lx \n", (int *)address);*/

}

/* ////////////////////////////////////////////////////////////////////

  Clear a system event. From RTX, may be useful for future
  implementation.

//////////////////////////////////////////////////////////////////// */
void  clearSystemEvent( unsigned event_id )
{

  disableInts();
  Event_Array[event_id] = 0;
  enableInts();

}

/* ////////////////////////////////////////////////////////////////////

  ClearEventArray()

//////////////////////////////////////////////////////////////////// */

void ClearEventArray(int index)
{
  Event_Array[index] = 0;

}
```

```
/* ///////////////////////////////////////////////////////////////////

tmemset : overwrites memory with given symbol

/////////////////////////////////////////////////////////////////// */

void * tmemset(void * pointer, int value, int len){


  int i;
  int * t = pointer;
  for (i = 0; i < len; i++){
   *(t + i) = value;

  }
  return(pointer);
}

/* ///////////////////////////////////////////////////////////////////

Function: mem_init
Input: none
Output: none
Purpose: initializes the memory queue

/////////////////////////////////////////////////////////////////// */


void mem_init(void ){

  int i;

  for(i = 0; i < 20; i++){

    Block_Array[i].block_num = i;
    Block_Array[i].block_address = 0;
    Block_Array[i].free = 1;
  }
}

/* ///////////////////////////////////////////////////////////////////

Function: block_malloc
Input: int i
Output: void pointer to a block of memory
```

```
Purpose: allocate assign a block of memory

/////////////////////////////////////////////////////////////////////// */
void * block_malloc(int i){

  const int mem_start = 0x1000;
  const int start = 0x0000;
  const int buffer_size = 0x0400;
  int buffer;
  const int block_size = 0x1000;
  int block;
  void * address = far_address(0x0000,0x0000);

  buffer = buffer_size * (i + 1);
  block = block_size *(i + 1) + (buffer);
  address = far_address(start, block);
  address = far_address(mem_start, buffer);

  return(address);
}

/* ///////////////////////////////////////////////////////////////////////

Function: tmalloc
Input: none
Output: void pointer to a block of memory
Purpose: allocate memory to each state

/////////////////////////////////////////////////////////////////////// */
void * tmalloc(void ){

  int i;

  for(i = 0; i < 20; i++){

  if(Block_Array[i].free){
    Block_Array[i].free = 0;
    break;
  }

}
return(block_malloc(i));
}
```

```
/* ////////////////////////////////////////////////////////////////////

Function: tfree
Input: void * address
Output: void pointer to a block of memory
Purpose: frees the allocated memory block

//////////////////////////////////////////////////////////////////// */

void * tfree(void * address){

  int i;
  Block_Array[i].free = 1;
  return(address);
}

/* ////////////////////////////////////////////////////////////////////

  State Machine Engine: Runs through State matrix to get the next state

//////////////////////////////////////////////////////////////////// */
void * StateSelect(void )
{
  int e,s, eflag;
  stateTypes Data_Path = DP_INIT;
  systemEvents Reattach_Event = REATTACH_EVENT;
  State * new_state = NULL;
  int old_state_num = currentState->state_num;

  eflag = 0;
  while ( new_state == NULL ){

    while( eflag < NUM_EVENTS){

      /*--- first find event ---*/
      if (Event_Array[eflag]){
            e = eflag;
        eflag = NUM_EVENTS;

        for (s = 0; s < NUM_STATES; s++){
          /*--- Then check the state ---*/
          if (Current_State_Array[s]) {

            /*--- If not in Data Path or not reattaching,
                then need to restart current state ---*/
```

```
        if(s > 3 && s < Data_Path && e != Reattach_Event){
          new_state = redefineState(State_Matrix[e][s]);
          break;
                 } /* if(s > 3 && s < Data_Path && e != Reattach_Event */
                 else{
          new_state = State_Matrix[e][s];
          break;
        }/* else */

            }/* if (Current_State_Array[s]) */

          }/* for s */

      }/* if (Event_Array[eflag]) */

    else eflag++;

  }/* while eflag */

 }/* while */

 ClearEventArray(e);

 if (new_state == NULL){
   new_state = currentState;
 }
 currentState = new_state;
 Current_State_Array[old_state_num] = 0;
 Current_State_Array[currentState->state_num] = 1;

 return currentState->curr_stack_addr;

}

/*//////////////////////////////////////////////////////////////////////////

     This function initializes the TCBEX's data structures

//////////////////////////////////////////////////////////////////////////*/
void  initStateSystem(void )
{
```

```
  tmemset(States, 0, sizeof(States));
  tmemset(Event_Array, 0, sizeof(Event_Array));
  tmemset(Current_State_Array, 0, sizeof(Current_State_Array));
  tmemset(State_Matrix, 0, sizeof(State_Matrix));
  mem_init();

  currentState = NULL;
  Current_State_Array[1] = 1;


  init_IntTraps();
  tprint(" State System Initialized ");


}

/*//////////////////////////////////////////////////////////////////////

        This function defines each state with a stack

//////////////////////////////////////////////////////////////////////*/
State * defineState(int ( *start_addr) (void * arg ),
                    void * argument,
                    unsigned stack_size,
                    int state_num,
                    char * state_name )

{
  void          * stack_addr;
  int           * stack_bottom;
  State         * state;

  state = &States[ state_num ];


  tmemset(state, NULL, sizeof(State));

  if ( (stack_addr = tmalloc()) == NULL )
    return NULL;


  stack_bottom = (int *)stack_addr;

  stack_bottom[ 0 ] = PROTECTION_PATTERN;
```

```
  stack_addr = (void  * )((char  *)stack_addr + stack_size - sizeof(int));

  state->stack_size = stack_size;
  state->state_num = state_num;
  state->start_addr = start_addr;
  state->argument = argument;
  state->init_stack_addr = stack_addr;

  state->curr_stack_addr = save_initregs
        ( stack_addr, start_addr, argument, state_num );


  return (&States[state_num]);

}

/*////////////////////////////////////////////////////////////////////////*/

        This function gives a state a new stack and address
     Space.
/*////////////////////////////////////////////////////////////////////////*/
State  *  redefineState(State *state )
{
  void  * stack_addr;
  void  * start_addr;
  void  * argument;
  int    * stack_bottom;
  int    state_num;
  unsigned stack_size;


  /*--- get the state's information  ---*/
  state->free = TRUE;
  start_addr = state->start_addr;
  state_num = state->state_num;
  argument = state->argument;


  /*--- free the space that it was using before --*/
  tfree(state->init_stack_addr);

  /*--- now give it new space and information ---*/
  if ( (stack_addr = tmalloc()) == NULL )
    return NULL;                          /* if tmalloc failed */
```

```
  stack_bottom = (int  *)stack_addr;
  stack_bottom[ 0 ] = PROTECTION_PATTERN;

  stack_addr = (void  * )((char  *)stack_addr + stack_size - sizeof(int));

  state->stack_size = stack_size;
  state->init_stack_addr = stack_addr;
  state->curr_stack_addr = save_initregs
        ( stack_addr, start_addr, argument, state_num );


  return state;
}

/*///////////////////////////////////////////////////////////////////////

   Called from terminateState_num. Frees the memory of the state pointed at.
   Modified to overwrite that portion of memory.

///////////////////////////////////////////////////////////////////////*/
int  terminateState(State * state )
{
 if ( state == currentState )
    currentState = NULL;


  state->free = TRUE;
  tfree(state->init_stack_addr);
  tmemset(state, NULL, sizeof(State));
  return(0);
}

/*///////////////////////////////////////////////////////////////////////

   Frees the memory of the corresponding state.  Only called from Task_Exit
   in regs.asm.  This function may be useful for redundancy when finishing
   or switching states

///////////////////////////////////////////////////////////////////////*/
int  terminateState_num(int state_num )
{

 terminateState(&States[ state_num ]);
  return (0);
 }
```

```
/*///////////////////////////////////////////////////////////////////////

 Left over from RTX. may be useful for future implementations

///////////////////////////////////////////////////////////////////////*/
extern int x_keyboardEventFct(event event_id, void * arg )
{

  int * kbd_buffPtr = far_address(0x0040, 0x001A);
  int kbd_buffer_status;

  disableInts();

  kbd_buffer_status = (kbd_buffPtr[ 0 ] != kbd_buffPtr[ 1 ]);

  enableInts();

  return kbd_buffer_status;

}
```

d. TCBE_Module.h

```
/*-------------------------------------------------------------------------
// Name: TCBE_Module.h
// Author: Jason Hackerson
// Date: 06 July 1998
// Function: Provide the header file with global information for the
//       entire TCBE System
//-----------------------------------------------------------------------*/


/* Structures used to pass data between modules and components on the TCBE and
the host computer*/

struct Data_Block_Record{
      char * Command;
      void * Data;
};
```

```
struct TCBE_Data_Block_Record{
      char * Command;
      struct Data_Block_Record Data_Block;
      int Data_Block_Size;
};

int TCBE_Initialize(void * argument);
int TCBE_Display(void * argument);
int DP_Path_Init(void * argument);
int DP_Path(void * argument);
```

e. TCBE_Module.c

```
/*----------------------------------------------------------------------------------
// Name: TCBE_Module.c
// Author: Jason Hackerson
// Date: 08 September 1998
// Purpose: Provides the TCBE Core modules for the TCBE. Also includes some
// stub functions for the Data Path Module
//--------------------------------------------------------------------------------*/

#include "tcbex.h"

/* One session number assigned for each session in use*/
int Session_Number;

/* particular state of the CPU and OS when halted by the TCBE; to be used when
TCBE has ability to hold client OS State*/

typedef struct  {
  int Client_OS_Size;
}Client_OS_Info_Record;

/* Tables to hold the Client OS state and the appropriate session number*/
struct Session_Info_Record{
      int Session_Num;
      Client_OS_Info_Record Client_OS_Info;
};

struct Session_Table_Record{
      int Current_Session_Number;
      struct Session_Info_Record* Session_Info;
};
```

```
int DP_Established = 0;


int TCBE_Load_Client_OS(void ){

  tprint("*** TCBE Loading Client OS ***");
  /* Code to load Client OS */
  DP_Established = 1;
  return(1);
}

int TCBE_Initialize(void * argument){
  tprint(" Initializing TCBE ");
  Transit();
  return (4);
}


int TCBE_Display(void * argument){
  tprint(" TCBE Display ");
  SAKTest();
  return (4);
}

int DP_Path_Init(void * argument){

  tprint(" Initializing Data Path ");
  /* Code to start data path */
  if(DP_Established){
        Transit();
  }
  else {
    tprint("*** Need to Load Client Operating System ***");

    if(TCBE_Load_Client_OS()){
      Transit();
    }
  }
  return(0);
}

int DP_Path(void * argument){
  tprint(" Data Path ");
  return (4);
}
```

f. TP_Module.h

```
/*-------------------------------------------------------------------------
// Name: TP_Module.h
// Author: Jason Hackerson
// Date: 09 July 1998
// Function: Provide the header file for the Trusted Module of the TCBE
//-----------------------------------------------------------------------*/


/* Only the functions that form states need to be public */
int TP_Start_Trusted_Path(void * argument);
int TP_Idle_Trusted_Path(void * argument);
int TP_Confirm_Trusted_Path(void * argument);
int TP_Login(void * argument);
int TP_Logout(void * argument);
int TP_Switch_Session_Level(void * argument);
```

g. TP_Module.c

```
/*-------------------------------------------------------------------------
// Name: TP_Module.c
// Author: Jason Hackerson
// Date: 10 September 1998
// Purpose: Provides the driver program for the TCBE. These functions are
// stub functions. The Trusted Path developer should put the Trusted Path code
// here.
//-----------------------------------------------------------------------*/

#include "tcbex.h"

enum Trusted_Path_Status_Type {Esatblished, Not_Established};
enum Trusted_Path_State_Type {Idle, Confirming, Initializing, Logging_Out,
                    Logging_In, Switching_Session_Level,
                    Clearing_Buffer};
typedef int Encryption_Key;
typedef int Data;


/* structures to keep track of where the TCBE is in security levels*/
```

```
struct Trusted_Path_Session_Info_Record{
  int Session_Number;
  Encryption_Key Session_Encryption_Key;
};

/* This structure expands if able to hold multiple session states;
// for initial implementation it will only have one entry*/

struct Trusted_Path_Session_Table_Record{
  int Current_Session_Number;
  struct Trusted_Path_Session_Info_Record* Trusted_Path_Session_Info;
};

/* Buffer holds requests*/

struct Trusted_Path_Buffer_Record{
  int Desired_Session_Level;
};

/* Self-explanatory*/

struct Current_User_Information_Record{
  char * Username;
  char * Password;
  Data Biometric_Data;
};

/* Global Values while operating in the Trusted Path*/
struct Trusted_Path_Session_Info_Record Trusted_Path_Session_Info;
struct Trusted_Path_Session_Table_Record Trusted_Path_Session_Table;
struct Trusted_Path_Buffer_Record Trusted_Path_Buffer_Record;
struct Current_User_Information_Record Current_User_Information;


int TP_Start_Session(int Session_Level){ return (2);}
int TP_Get_Session_Key(void ){ return (2);}


int TP_Start_Trusted_Path(void * argument){

  int result = 0;
  result = 1;
  tprint(" Starting Trusted Path ");
  /* Code to initialize the trusted path */
  Transit();
```

```
    return(result);

}

int TP_Idle_Trusted_Path(void * argument){

    commandTypes Command;
    static int count = 1;

    /* Code to display Trusted Path Menu or start screen */
    /* Code to allow the user to input choice or command */

    tprint(" Trusted Path Display ");

    if (count == 0){Command = CONFIRM;}
    if (count == 1){Command = LOGIN;}
    if (count == 2){Command = LOGOUT;}
    if (count == 3){Command = REATTACH;}
    if (count == 4){Command = SWITCH_SESSION;}

    if(Command == CONFIRM){
      SAKTest();
    }
    else if(Command == LOGIN){
      Login();
    }
    else if(Command == LOGOUT) {
      Logout();
    }
    else if(Command == REATTACH){
      Reattach();
    }
    else Switch_Session();


    count++;
    return (1);
}

int TP_Confirm_Trusted_Path(void * argument) {

    tprint (" Confirming Trusted Path ");
    /* Code to confirm the Trusted Path with the
      high assurance server  */
    Transit();
```

```
return (2);
}

int TP_Login(void * argument){

  tprint(" Logging in to High Assurance Server ");
  /* Code to login to high assurance server */
  Transit();
  return (2);

}


int TP_Logout(void * argument){
  tprint(" Logging out of High Assurance Server ");
  /* Code to log out of high assurance server */
  Transit();
  return (2);

}
```

h. Stddefs.asm

```
    ;''''''''''''''''''''''''''''''''''''''''''''''''''''''''';
    ; Standard Definitions.;
    ;--------------;

    ; (C) Copyright, 1991.  Mike Podanoffsky   ;
    ;    All Rights Reserved.                  ;
    ;.....Macros developed by original author  ;

zero        equ 0
one         equ 1
two         equ 2
minusOne    equ -1

no          equ 0
yes         equ 1

false       equ 0
true        equ 1
```

```
__pointer     equ 0
__segment     equ 2


;----------------------------------
; Dos Function Codes; Not used by TCBEX
;----------------------------------
ExitProgram                equ 00h

SetDiskTransferAddr        equ 1Ah
GetDiskTransferAddr        equ 2Fh
SetIntVector               equ 25h
GetIntVector               equ 35h
TerminateProcess           equ 4ch


;----------------------------------
; Flag Register Values
;----------------------------------
_flags_of          equ 0800h
_flags_df          equ 0400h
_flags_if          equ 0200h
_flags_tf          equ 0100h
_flags_sf          equ 0080h
_flags_zf          equ 0040h
_flags_af          equ 0010h
_flags_pf          equ 0004h
_flags_cf          equ 0001h

    ;''''''''''''''''''''''''''''''''''''''';
    ;                                       ;
    ; Entry Macro.                          ;
    ;.......................................;

entry  macro label

label  proc far
_ARGP  = 6
_ARGT  = 0
    push bp
    mov bp,sp
    endm
```

```
        ;''''''''''''''''''''''''''''''''''''''''''''';
        ;                                ;
        ; alias Macro.                   ;
        ;................................;

alias   macro label

label   proc far
label   endp
        endm

        ;''''''''''''''''''''''''''''''''''''''''''''';
        ;                                ;
        ; Return Macro.                  ;
        ;................................;

return  macro nbytes
        mov sp,bp
        pop bp
   ifidn <nbytes>,<pascal>
           ret _ARGP-6
           else
           ret nbytes
           endif
        endm

        ;''''''''''''''''''''''''''''''''''''''''''''';
        ;                                          ;
        ; Define Double Argument.                  ;
        ;..........................................;

darg    macro label

label   = _ARGP
_ARGP   = _ARGP+4                          ;long
           chklabel label
        endm

        ;''''''''''''''''''''''''''''''''''''''''''''';
        ;                                ;
        ; Define Argument.               ;
        ;................................;

arg     macro label

label   = _ARGP                        ;set argument.
_ARGP   = _ARGP+2
           chklabel label
        endm
```

```
        ;""""""""""""""""""""""""""""""""""""""""""";
        ;                                          ;
        ; Temporary Storage.                       ;
        ;..........................................;

targ    macro label,reg

_ARGT   = _ARGT-2
label   = _ARGT                         ;set argument.
pushedreg = no

   ifb <reg>
      push ax
      pushedreg = yes
      endif

   ifnb <reg>
    irp treg,<ax,bx,cx,dx,si,di,es,ds,f>
     ifidn <reg>,<treg>
      tpush reg
      pushedreg = yes
      endif
      endm
      endif

    ife pushedreg-no                    ;if not pushed
      push ax
      mov word ptr label [bp],reg           ;really a value.
      endif
         chklabel label
         endm

        ;""""""""""""""""""""""""""""""""""""""""""";
        ;                                          ;
        ; Temporary Double Arg Storage.            ;
        ;..........................................;

tdarg   macro label,reg,seg

_ARGT   = _ARGT-4
label   = _ARGT                         ;set argument.
pushedreg = no
pushedseg = no

   ifb <reg>
```

```
    push ax
    push ax
    pushedreg = yes
    pushedseg = yes
    endif

  ifnb <reg>
   irp treg,<ax,bx,cx,dx,si,di,es,ds,f>
    ifidn <reg>,<treg>
     tpush reg
     pushedreg = yes
     endif
     endm
     endif

   ife pushedreg-no                       ;if not pushed
     push ax
     push ax
     mov word ptr label [bp],reg           ;really a value.
     mov word ptr label [bp+2],zero          ;32 bits.
     pushedreg = yes
     pushedseg = yes
     endif

  ifnb <seg>
   irp tseg,<es,ds>
    ifidn <seg>,<tseg>
     tpush seg
     pushedseg = yes
     endif
     endm
     endif

   ife pushedseg-no
     push ax
     endif
        chklabel label
     endm

     ;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''.
     ;                      ;
     ; Check for Bad Label  ;
     ;......................................;

chklabel    macro label
```

```
  ifidn <label>,<offset>
        error "bad label"
        endif

  ifidn <label>,<org>
        error "bad label"
        endif

  ifidn <label>,<ptr>
        error "bad label"
        endif

  ifidn <label>,<end>
        error "bad label"
        endif

  ifidn <label>,<endp>
        error "bad label"
        endif

  ifidn <label>,<even>
        error "bad label"
        endif

  ifidn <label>,&lt;segment&gt;
        error "bad label"
        endif

        endm

      ;'''''''''''''''''''''''''''''''''''''''''''''';
      ;  Define Double Argument. ;
      ;..............................................;

getdarg  macro seg,reg,label
loadedflag  = no

 ifidn <seg>,<es>
     les reg,dword ptr label [bp]
     loadedflag = yes
     endif

 ifidn <seg>,<ds>
     lds reg,dword ptr label [bp]
     loadedflag = yes
```

```
        endif

   ife loadedflag-no
        mov reg,word ptr label [bp]
        mov seg,word ptr label [bp+2]
        endif
        endm

        ;''''''''''''''''''''''''''''''''''''''''''''''''';
        ;                                  ;
        ; Define Argument.                 ;
        ;..................................;

getarg  macro reg,label
        mov reg,word ptr label [bp]
        endm

        ;''''''''''''''''''''''''''''''''''''''''''''''''';
        ;                                  ;
        ; Store Double Argument.           ;
        ;..................................;

stordarg  macro label,reg,seg
        mov word ptr label [bp],reg
        mov word ptr label [bp+2],seg
        endm

        ;''''''''''''''''''''''''''''''''''''''''''''''''';
        ;                                  ;
        ; Store Argument.                  ;
        ;..................................;

storarg macro label,reg
        mov word ptr label [bp],reg
        endm

        ;''''''''''''''''''''''''''''''''''''''''''''''''';
        ;                                  ;
        ; Save Registers.                  ;
        ;..................................;

saveRegisters macro

        push ax
        push bx
        push cx
        push dx
        push si
        push di
```

```
    push bp

    push ds
    push es
    endm

    ;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''';
    ;                                 ;
    ; Restore Registers.              ;
    ;.................................;

restoreRegisters macro
    pop es
    pop ds

    pop bp
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    endm

    ;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''';
    ;                                 ;
    ; Save Segments.                  ;
    ;.................................;

saveSegments macro reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg0

    push ds
    push es

  ifnb <reg1>
    tpush reg1
    endif
  ifnb <reg2>
    tpush reg2
    endif
  ifnb <reg3>
    tpush reg3
    endif
  ifnb <reg4>
    tpush reg4
    endif
  ifnb <reg5>
```

```
        tpush reg5
        endif
    ifnb <reg6>
        tpush reg6
        endif
    ifnb <reg7>
        tpush reg7
        endif
    ifnb <reg8>
        tpush reg8
        endif
    ifnb <reg9>
        tpush reg9
        endif
    ifnb <reg0>
        tpush reg0
        endif
        endm

        ;''''''''''''''''''''''''''''''''''''''''''''''''''';
        ;                                                  ;
        ;  Restore Segments.    ;
        ;..................................;

restoreSegments macro reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg0

    ifnb <reg1>
        tpop reg1
        endif
    ifnb <reg2>
        tpop reg2
        endif
    ifnb <reg3>
        tpop reg3
        endif
    ifnb <reg4>
        tpop reg4
        endif
    ifnb <reg5>
        tpop reg5
        endif
    ifnb <reg6>
        tpop reg6
        endif
    ifnb <reg7>
        tpop reg7
```

```
    endif
 ifnb <reg8>
    tpop reg8
    endif
 ifnb <reg9>
    tpop reg9
    endif
 ifnb <reg0>
    tpop reg0
    endif

    pop es
    pop ds
    endm

    ;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''';
    ;                                              ;
    ; This Segment                                 ;
    ;..............................................;

thisSegment macro reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg0

 ifnb <reg1>
    push cs
       pop reg1
    endif
 ifnb <reg2>
    push cs
       pop reg2
    endif
 ifnb <reg3>
    push cs
       pop reg3
    endif
 ifnb <reg4>
    push cs
       pop reg4
    endif
 ifnb <reg5>
    push cs
       pop reg5
    endif
 ifnb <reg6>
    push cs
       pop reg6
    endif
```

```
ifnb <reg7>
   push cs
     pop reg7
   endif
ifnb <reg8>
   push cs
     pop reg8
   endif
ifnb <reg9>
   push cs
     pop reg9
   endif
ifnb <reg0>
   push cs
     pop reg0
   endif
   endm

   ;'''''''''''''''''''''''''''''''''''''''''''''''''''''';
   ; Call Dos                                    ;
   ;......................................................;

callDos macro fctvalue,lowvalue

  ifnb <fctvalue>
     mov ah,fctvalue
     endif

  ifnb <lowvalue>
     mov al,lowvalue
     endif

     int 21h
     endm

   ;'''''''''''''''''''''''''''''''''''''''''''''''''''''';
   ; Set Interrupt Vector.                       ;
   ;......................................................;

setIntTrap macro intvector, newvector, savevector

     callDos GetIntVector, intvector
     mov word ptr [savevector + 2 ], es
     mov word ptr [savevector ], bx
```

```
mov dx,seg newvector
mov ds,dx
mov dx,offset newvector
callDos SetIntVector, intvector

endm
```

i. Regs.asm

```
title "tcbex register support"
page ,132

;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''';
; Interrupt/ Register Support for TCBEX
;-----------------------------;
; (C) Copyright, 1991. Mike Podanoffsky
;     All Rights Reserved
; Technical questions: 508/ 454-1620.
; Modified by Jason Hackerson, September 1998
;....................................................;

.model large
include stddefs.asm


reg_supp segment para public 'CODE'
assume cs:reg_supp,ds:reg_supp,es:reg_supp,ss:reg_supp


public _tprint                  ; provides simple print function
public _ischeduler              ; iret scheduler call
public _scheduler               ; far call scheduler call
public _keyboardEventFct        ; keyboard event function

public _condenableInts          ; allow ints on cond
public _enableInts              ; enable interrupts
public _disableInts             ; disable interrupts


public _save_initregs           ; used to init a task's stack
public _init_IntTraps           ; initial state interrupt traps
public _restore_IntTraps        ; restore interrupt traps
```

```
public _SaveRegisters
public _RestoreRegisters

extrn _terminateState_num:proc
extrn _StateSelect:proc
extrn _saveCurrentState_Stack:proc
extrn _returnState_StackFrame:proc

extrn _waitKbdEvent:proc
extrn _setKbdEvent:proc
extrn _setSAKEvent
extrn _printa:proc


;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''';
; Flags/ Storage Definitions
;...........................................;


schedulerPending:          dw 0
tcbex_DataSegment:         dw 0


_originalDosTrap:          dd 0
_originalTickInt:          dd 0
_originalKbdSvcTrap:       dd 0
_originalKbdHdwTrap:       dd 0
_originalScrnTrap:         dd 0

OldInt9      dw            ?
OldInt8      dw            ?
OldInt10     dw            ?
OldInt16     dw            ?
OldInt21     dw            ?

SAK_Scan_Code              equ    1fh     ;alt-S
AltBit                     equ    8
KbdFlags                   equ    <byte ptr ds:[17h]>

total_ints:                dw 0


;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''';
; Macro Definitions.
;...................................;
```

```
_Int    macro loc

        pushf
        call dword ptr cs:[loc]          ; keep rentrancy problems down.

        endm



        ;''''''''''''''''''''''''''''''''''''''''''''''''';
        ;                                       ;
        ; save_initregs                         ;
        ;---------------------------------------;
        ;
        ; Initializes stack for the state
        ;  void * save_initregs
        ;      ( void far * stack,
        ;        void far * start_addr,
        ;        void far * argument,
        ;        int task_id );
        ;                                   ;
        ;...............................................;

    entry _save_initregs

    darg __stack
    darg __start_addr
    darg __argument
    arg  __task_id

;-----------------------------------
; order of items stored to task's stack
;-----------------------------------
_task_id        equ    40
_argument       equ    36              ; void far *
_taskExit       equ    32              ; task exit address

_start_addr     equ    28              ; start address
_pushf          equ    26                  ; push flag emulation
_push_ax        equ    24              ; ax
_push_bx        equ    22              ; bx
_push_cx        equ    20              ; cx
_push_dx        equ    18              ; dx
_push_si        equ    16              ; si
_push_di        equ    14              ; di
```

```
_push_bp      equ   12                    ; bp

_push_ds      equ   10                    ; ds
_push_es      equ   08                    ; es

_push_dta     equ   04                    ; dta
_push_stackf  equ   00                    ; null stack frame pointer

FLAGS_VALUE    equ    0246h               ; ei zr nc


;---------------------------------
; execution starts here.
;---------------------------------
    saveSegments

    push ds
    pop es                          ; insure es: = ds:
    getdarg ds,bx,__stack

    sub bx,128                      ; disk transfer address
    push bx
    push ds                         ; ds:bx

    sub bx,_task_id+2               ; total arguments we'll store

;---------------------------------
; task stack will contain task_id, return address, ...
;---------------------------------

    getarg ax, __task_id
        mov word ptr _task_id[bx],ax

    getdarg dx, ax, __argument
    mov word ptr _argument+2[bx],dx
    mov word ptr _argument[bx],ax

    mov word ptr _taskExit+2[bx], seg taskExit
    mov word ptr _taskExit[bx], offset taskExit

    mov word ptr _pushf[bx], FLAGS_VALUE   ; flags.

    getdarg dx, ax, __start_addr
    mov word ptr _start_addr+2[bx],dx
    mov word ptr _start_addr[bx],ax
```

```
;--------------------------------
; now save on stack ALL registers
;--------------------------------
    xor ax,ax
    mov word ptr _push_ax[bx],ax
    mov word ptr _push_bx[bx],ax
    mov word ptr _push_cx[bx],ax
    mov word ptr _push_dx[bx],ax
    mov word ptr _push_si[bx],ax
    mov word ptr _push_di[bx],ax
    mov word ptr _push_bp[bx],ax

    mov word ptr _push_ds[bx],ss
    mov word ptr _push_es[bx],ss

    pop word ptr _push_dta[bx]          ; seg  (dta)
    pop word ptr _push_dta + 2[bx]      ; address


    xor ax,ax
    mov word ptr _push_stackf[bx],ax      ; seg (stack frame)
    mov word ptr _push_stackf+ 2[bx],ax   ; address

;--------------------------------
; return adjusted stack address
;--------------------------------
    mov dx,ds
    mov ax,bx                     ; dx:ax contain remaining stack.

    restoreSegments
    return pascal

_save_initregs     endp

    ;''''''''''''''''''''''''''''''''''''''';
    ; scheduler
    ;--------------------------;
    ;
    ; No parameters.
    ;                                    ;
    ; Will suspend current task, schedule next task.
    ; See related function: ischedule() for isr calls.
    ;                                    ;
    ;......................................;
```

```
_scheduler        proc far

        pushf                           ; save flags

        SaveRegisters

        push bx
        push es

        cli
        call _returnState_StackFrame
        push dx                         ; save current stack frame.
        push ax

        mov dx,ss
        mov ax,sp
        push dx
        push ax
        call _saveCurrentState_Stack
        sti
        xor dx,dx


;-------------------------------
; stack switch
;-------------------------------
scheduler_32:

        call _StateSelect
        mov cx,ax
        or cx,dx                        ; NULL value ?

        jnz scheduler_36                ; if new state then switch
        int 3
        jmp scheduler_38                ; else restore

scheduler_36:
        cli                             ; switch state stacks.
        mov ss,dx
        mov sp,ax

;-------------------------------
; if return, stack switch
```

```
;-----------------------------
scheduler_38:
      call _saveCurrentState_Stack    ; restore stack frame.
      sti                             ; ok ints now


      ; restore the registers
      ; first six pops get the stack pointer to the correct spot on the
      ; state's stack
      pop ax
      pop ax

      pop ax
      pop ax

      pop ax
      pop ax

      pop bp
      pop di
      pop si
      pop dx
      pop cx
      pop bx
      pop ax

      popf                      ; this will restore ints
      mov al, 'X'               ; marker for scheduler switching
      call PutcBIOS
      xor ax, ax
      xor dx, dx
      ret far

_scheduler       endp

      ;''''''''''''''''''''''''''''''''''''''''''''';
      ; ischeduler
      ;-------------------------;
      ;
      ; No parameters.
      ;
      ; Will suspend current task, schedule next task.
      ; Performs an iret.  Some ISRs may find this call
      ; preferable.
      ;
```

```
;................................................;

_ischeduler      proc far

        call _scheduler
        iret
_ischeduler      endp


        ; taskExit()
        ;----------------------------;
        ;
        ; This process is called whenever a task terminates.
        ; Calls scheduler to release task.
        ;
        ;................................................

taskExit      proc far
        mov bp,sp                       ; didn't get here through a call
        mov dx,word ptr [bp]            ; should be task_id

; switch to temp stack, then ...
        cli
        mov ds, word ptr cs:[tcbex_DataSegment]
        mov ax,seg temp_stack
        mov ss,ax
        mov sp,offset temp_stack

        sti
        push dx                         ; copy to stack
        call _terminateState_num        ; terminate task

taskExit_08:
        call _scheduler
        jmp taskExit_08

taskExit      endp


        ;''''''''''''''''''''''''''''';
        ; enableInts()
        ;----------------------------;
        ;
        ; Enables interrupts for C programs.
        ;
```

```
;..................................................

_enableInts     proc far

        sti
        ret

_enableInts     endp

        ;''''''''''''''''''''''''''''''''''''''''''''''';
        ; disableInts()
        ;-----------------------------;
        ;                                              ;
        ; Disables interrupts for C programs.
        ;                                              ;
        ;.................................................;

_disableInts    proc far

        pushf
        pop ax                          ; copy flags to ax.
        and ax,_flags_if
        jz disableInts_08               ; if ints were disabled -->
        mov ax,1                        ; if ints are enabled.

disableInts_08:
        cli
        ret

_disableInts    endp

        ;''''''''''''''''''''''''''''''''''''''''''''''';
        ; condenableInts()
        ;-----------------------------;
        ;
        ; Conditional enables interrupts. Use disableInts to
        ; disable interrupts. This is a convenience function.
        ;
        ;..................................................

    entry _condenableInts
        or ax,ax
        jz condenableInts_08
        sti
```

```
condenableInts_08:
        return

_condenableInts  endp



        ;'int 08'''''''''''''''''''''''''''''''''''';
        ; TimerTick
        ;---------------------------;
        ;
        ; This ISR is called every clock tick (int 08)
        ; If the current clock value at interrupt equals or
        ; passes the next required clock value, the scheduler
        ; is called to determine what action is required next.
        ;                                              ;
        ;..............................................;

TimerTick       proc far

        push ax
        push bx
        push dx
        push ds

        _Int _originalTickInt            ; emulate int call
        inc word ptr cs:[ total_ints ]

;--------------------------------
; see if min timer value expired.
;--------------------------------
        mov ax,40h
        mov ds,ax
        mov bx, 6Ch

        mov ax,word ptr [ bx ]
        mov dx,word ptr [ bx + 2 ]

        push cs
        pop ds                           ; set ds to current.
        jg TimerTick_40

        sti
        xor ax,ax
```

```
        saveRegisters
        mov ds, word ptr cs:[tcbex_DataSegment]
        call _scheduler                    ; see if task time slice
        restoreRegisters

TimerTick_40:
        pop ds
        pop dx
        pop bx
        pop ax

        iret

TimerTick       endp

        ;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''';
        ; restore_IntTraps()
        ;--------------------------------------;
        ;
        ; Restores interrupt traps before exit.
        ; Not used for the TCBEX.
        ;.....................................................

_restore_IntTraps proc far

        lds dx,dword ptr cs:[_originalDosTrap]
        callDos SetIntVector, 21h            ; restore vector to Int21.

        lds dx,dword ptr cs:[_originalKbdHdwTrap]
        callDos SetIntVector, 09h

        lds dx,dword ptr cs:[_originalScrnTrap]
        callDos SetIntVector, 10h

        lds dx,dword ptr cs:[_originalKbdSvcTrap]
        callDos SetIntVector, 16h

        lds dx,dword ptr cs:[_originalTickInt]
        callDos SetIntVector, 08h

        ret
_restore_IntTraps endp
```

```
;''''''''''''''''''''''''''''''''''''''''''''''''';
; Int21()
;---------------------------;
;
; Int 21 traps to here.  We can detect DMA address
; change, terminate process, and whether we are
; entering/exiting DOS.
;                                              ;
;..............................................;

Int21   proc far

      cmp ah,ExitProgram
      jz Int21_12
      cmp ah,TerminateProcess
      jnz Int21_20

Int21_12:
      push ax                               ; save return code.

      lds dx,dword ptr cs:[_originalDosTrap]
      callDos SetIntVector, 21h             ; restore vector to Int21.
      call _restore_IntTraps
      pop ax

Int21_20:
      _Int _originalDosTrap                 ; emulate int call
      ret 2                                 ; pass through our own status.

Int21   endp

            ;''''''''''''''''''''''''''''''''''''''''''''''''';
            ;
            ;       SetCmd - this procedure communicates with the 8042
            ;       processor in the keyboard.  Code borrowed from Randall
            ;       Hyde's Art of Assembly
            ;
            ;''''''''''''''''''''''''''''''''''''''''''''''''';


SetCmd      proc   near
            push   cx
            push   ax
            cli
```

```
                xor     cx, cx

                mov al, 'F'
                call PutcBIOS
                xor ax, ax

Wait4Empty:     in      al, 64h                 ;Read Keyboard status register
                test    al, 10b                 ;Input Buffer full?
                loopnz Wait4Empty               ;if so wait until empty

                ;send command to the 8042

                pop     ax
                out     64h, al
                sti
                pop     cx
                ret
SetCmd                  endp


        ;''''''''''''''''''''''''''''''''''''''''''';
        ;                                          ;
        ; Int16()
        ;---------------------------;
        ;
        ;
        ; Int 16 is used to read the keyboard. It needs to
        ; trap to a waitKbdEvent() function which sets the
        ; KBD event flag in the current task and exits to the
        ; scheduler.
        ;
        ;
        ;...............................................

Int16   proc far
        or ah,ah                        ; is it a keyboard wait call ?
        jnz Int16_22                    ; no, go ahead and execute -->

        SaveRegisters
        mov ah,1
        int 16h                         ; we'll call ourselves
        jnz Int16_20                    ; if key available, no need waiting -->

        mov ds, word ptr cs:[tcbex_DataSegment]
        call _waitKbdEvent              ; wait on kbd event.

Int16_20:
```

```
    RestoreRegisters

Int16_22:
    _Int _originalKbdSvcTrap

    ret 2                        ; pass through our own status.


Int16   endp

    ;''''''''''''''''''''''''''''''''''''''''''''''''''''''';
    ; Int09()
    ;-----------------------------;
    ;
    ; Int 09 is the keyboard interrupt.  If a key is
    ; detected and saved by the ROM BIOS it will force
    ; the setKbdEvent and call the scheduler to evaluate
    ; task priorities.  This code has been modified by
    ; Jason Hackerson. Original RTX code has been commented
    ; out.
    ;..................................................;

Int09   proc far

    _Int _originalKbdHdwTrap         ; do normal kbd duties.

    ;push ax
    ;push bx
    ;push ds

    push dx
    push ax
    push cx


    mov ax,40h
    mov ds,ax                    ; look at bios kbd area
    ;mov bx, 1Ah

    mov al, 40h
    call SetCmd
    cli

    xor cx, cx
Wait4Data:
```

```
        in      al, 64h
        test    al, 10b
        loopz           Wait4Data
        in      al, 60h
        cmp     al, SAK_Scan_Code               ;is it the alt-S key?
        jne     OrigInt9

        ;if it is the alt-S key then eat the S and do not let it through

        mov     al, 0aeh                        ;reenable keyboard
        call    SetCmd
        mov     al, 20h                         ;send EOI
        out     20h, al                         ; to the 8259APIC
        pop     cx
        pop     ax
        pop     ds


        ;mov ax, word ptr [bx]
        ;cmp ax, word ptr [bx + 2]
        ;sti
        ;jz Int09_20

        SaveRegisters
        mov ds, word ptr cs:[tcbex_DataSegment]
        call _setSAKEvent                       ; say Secure Attention Key event occurred.
        call _scheduler
        RestoreRegisters

Int09_20:
        ;pop ds
        ;pop bx
        ;pop ax
        ;iret

OrigInt9:
        mov     al, 0aeh                        ; Reenable keyboard
        call    SetCmd
        pop     cx
        pop     ax
        pop     ds
        jmp     cs:OldInt9


Int09   endp
```

```
;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''.
;                                  ;
; KeyboardEventFct
;---------------------------;
;
; Sample of an event fct.  This returns true (non-zero)
; when keyboard data is available; false (zero) when no
; keyboard data is available.
;
;..................................................;

entry _keyboardEventFct

    push bx
    push ds
    mov ax,40h
    mov ds,ax                          ; look at bios kbd area
    mov bx,1Ah

    cli
    mov ax,word ptr [ bx ]             ; 1Ah
    cmp ax,word ptr [ bx + 2 ]         ; 1Ch
    sti

    mov ax, 0                          ; false if no input available.
    jz keyboardEventFct_08             ; if zero, no keys pending ->

    mov ax, 1                          ; true if input available.

keyboardEventFct_08:
    pop ds
    pop bx
    return

_keyboardEventFct  endp

;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''.
;                                  ;
; Int10()
;---------------------------;
;
; This would handle any program to Int 10 calls.  There
; are no screen special considerations unless you need
; to support different windows per task.
;
;..................................................
```

```
Int10   proc

    _Int _originalScrnTrap

    iret
Int10   endp

Set_IntTrap         macro intno, newint, oldint

            cli
            mov     ax, 0h
            mov     es, ax
            mov     ax, es:[intno*4]
            mov     word ptr oldint, ax
            mov     ax, es:[intno*4+2]
            mov     word ptr oldint+2, ax
            mov     es:[intno*4], offset newint
            mov     es:[intno*4+2],cs
            sti
endm

    ;''''''''''''''''''''''''''''''''''''''''''''''''''';
    ;                                                   ;
    ; init_IntTraps()
    ;-------------------------------------------;
    ;
    ; Init int traps for whole system.  Called by
    ; initTaskSystem().
    ;
    ;...................................................;

_init_IntTraps  proc far

    saveSegments

    mov     ax, ds
    mov     ds, ax

    push cs
    pop ds

    Set_IntTrap     9, Int09,   _originalKbdHdwTrap

    Set_IntTrap     10, Int10,  OldInt10
```

```
        Set_IntTrap     16, Int16,     _originalKbdSvcTrap

        Set_IntTrap     08, TimerTick, OldInt8

        Set_IntTrap     21, Int21,     OldInt21


        restoreSegments
        ret
_init_IntTraps   endp


_SaveRegisters  proc far

        push bp
        sub sp,10
        mov bp,sp                               ; points to bottom of table

        push word ptr [bp+14 ]                  ; return address
        push word ptr [bp+12 ]
        push word ptr [bp+10 ]                  ; bp pushed on call

        mov word ptr [bp+00 ], ax
        mov word ptr [bp+02 ], bx
        mov word ptr [bp+04 ], cx
        mov word ptr [bp+06 ], dx
        mov word ptr [bp+08 ], si
        mov word ptr [bp+10 ], di
        mov word ptr [bp+12 ], es
        mov word ptr [bp+14 ], ds


        pop bp
        ret

_SaveRegisters  endp


_RestoreRegisters proc far

        push bp
        mov bp,sp                               ; points to bottom of table
        add bp,6

        mov ax, word ptr [bp+00 ]
```

```
        mov bx, word ptr [bp+02 ]
        mov cx, word ptr [bp+04 ]
        mov dx, word ptr [bp+06 ]
        mov si, word ptr [bp+08 ]
        mov di, word ptr [bp+10 ]
        mov es, word ptr [bp+12 ]
        mov ds, word ptr [bp+14 ]

        pop bp
        ret 16

_RestoreRegisters endp

_tprint proc    far
                push    bp
                mov     bp, sp
                push    ax
                push    es
                push    bx
;
                les     bx, [bp + 6]            ;Get return address
                jmp     short TestZero
;
PrintLoop:      call    PutcBios
                inc     bx

TestZero:       mov     al, es:[bx]
                cmp     al, 0
                jnz     PrintLoop
;
                inc     bx
                mov     [bp + 6], bx
                pop     bx
                pop     es
                pop     ax
                pop     bp
                ret
_tprint endp

PutcBIOS        proc    near
                push    ax
                mov     ah, 14
                int     10h
                pop     ax
                ret
```

```
PutcBIOS      endp


reg_supp      ends

        ;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''';
        ; temporary stack
        ;.......................................................;

t_stack        segment para 'DATA'
               dw 4000 dup( 0 )
temp_stack     dd 0
t_stack        ends

        end
```

j. Makefile

```
#
# Borland C++ IDE generated makefile
# Generated 8/17/98 at 10:43:32 AM
#
.AUTODEPEND


#
# Borland C++ tools
#
IMPLIB  = Implib
BCCDOS  = Bcc +BccDos.cfg
TLINK   = TLink
TLIB    = TLib
TASM    = Tasm
#
# IDE macros
#
#
# External tools
#
Assembler = tasm.exe  # IDE Command Line: $TASM /mx
```

```
#
# Options
#
IDE_LinkFLAGSDOS = -LD:\BC5\LIB
IDE_BFLAGS =
LinkerLocalOptsAtDOS_tcbedexe = -c -Tde
ResLocalOptsAtDOS_tcbedexe =
BLocalOptsAtDOS_tcbedexe =
CompInheritOptsAt_tcbedexe = -ID:\BC5\INCLUDE
LinkerInheritOptsAt_tcbedexe = -x
LinkerOptsAt_tcbedexe = $(LinkerLocalOptsAtDOS_tcbedexe)
ResOptsAt_tcbedexe = $(ResLocalOptsAtDOS_tcbedexe)
BOptsAt_tcbedexe = $(BLocalOptsAtDOS_tcbedexe)

#
# Dependency List
#
Dep_tcbe = \
  tcbe.exe

tcbe : BccDos.cfg $(Dep_tcbe)
  echo MakeNode

Dep_tcbedexe = \
  regs.obj\
  tp_mod~1.obj\
  tcbe.obj\
  tcbex.obj\
  tcbe_m~1.obj

tcbe.exe : $(Dep_tcbedexe)
  $(TLINK)  @&&|
 /v $(IDE_LinkFLAGSDOS) $(LinkerOptsAt_tcbedexe)
$(LinkerInheritOptsAt_tcbedexe) +
D:\BC5\LIB\c0l.obj+
regs.obj+
tp_mod~1.obj+
tcbe.obj+
tcbex.obj+
tcbe_m~1.obj
$<,$*
D:\BC5\LIB\cl.lib

|
```

k. Batch File

j.bat

```
tasm /mx regs.asm
bcc -c -ml tcbex.c
bcc -c -ml tp_mod~1.c
bcc -c -ml tcbe_m~1.c
bcc -c -ml tcbe.c
make
copy d:\bc5\bin\hcode\tcbex\tcbe.exe a:
```

LIST OF REFERENCES

[1] *Department of Defense Trusted Computer System Evaluation Criteria,* DoD 5200.28-STD, December 1985.

[2] *A Guide to Understanding Object Reuse in Trusted Systems,* National Computer Security Center, NCSC-TG-018 VERSION-1, July 1992.

[3] *A Guide to Understanding Data Remanence in Automated Information Systems,* National Computer Security Center, NCSC-TG-025 VERSION-2, September 1991.

[4] *Software Systems Construction with Examples in Ada,* Sanden, Bo, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[5] *Object-Oriented Modeling and Design*, Rumbaugh, James; Blaha, Micheal; Premerlani; William, Eddy; Frederick, Lorensen, William. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1991.

[6] "IBM Network Computer Reference Platform White Paper", International Business Machines Corporation, USA, June 1997. http://www.chips.ibm.com/nc/whitepaper.html

[7] *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* Intel Corporation, USA, 1997.

[8] "Primary Executive", National Aeronautics and Space Administration. http://www.dfrc.nasa.gov/Projects/F18SRA/ARTS/sra-001/execp.html

[9] "Using Encryption for Authentication in Large Networks of Computers", Needham, R.; Schroeder, M. *Communications of the ACM,* December 1978.

[10] "Kerberos Authentication and Authorization System", Miller, S.; Neuman, B.; Schiller, J.; Saltzer, J. *Section E.2.1, Project Athena Technical Plan,* M.I.T. Project Athena, Cambridge, MA, 27 October 1998.

BIBLIOGRAPHY

A Guide to Understanding Identification and Authentication in Trusted Systems, National Computer Security Center, NCSC-TG-017 VERSION-1, September 1991.

A Guide to Understanding Design Documentation in Trusted Systems, National Computer Security Center, NCSC-TG-007 VERSION-1, 2 October 1988.

A Guide to Understanding Security Modeling in Trusted Systems, National Computer Security Center, NCSC-TG-010 VERSION-1, October 1992.

"A Multilevel File System for High Assurance", Irvine, Cynthia E. Computer Science Department, Naval Postgraduate School, Proceeding of the IEEE Symposium on Security and Privacy, Oakland, pp. 78-87, May 1995.

"High Assurance Multilevel Services for Off-the-Shelf Workstation Applications", Irvine, Cynthia E., Anderson, James P., Hackerson, Jason X., Draft, 27 February 1998.

"The Architecture of a Distributed Trusted Computing Base", Fellows, Jon; Hemenway, Judy; Kelem, Nancy; Romero, Sandra. Unisys, Santa Monica, CA.

"A Secure and Reliable Bootstrap Architecture", Arbaugh, William A.; Farber, David J.; Smith, Jonathan M. University of Pennsylvania, Distributed Systems Laboratory, Philadelphia, IEEE, pp. 65-71, November 1997.

"Dyad: A System for Using Physically Secure Coprocessors", Tygar, J.D.; Yee, Bennet School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 4 May 1991.

Assembly Language for the IBM-PC, Irvine, Kip R. Macmillan Publishing Co., New York, NY, 1990.

Mastering Turbo Assembler, Swan, Tom. Hayden Books, Carmel, IA, 1989.

The Indispensable PC Hardware Book, Messmer, Hans-Peter. Addison Wesley Longman Ltd. England, 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library..2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Director, Training and Education...1
   MCCDC, Code C46
   1019 Elliot Rd.
   Quantico, Virginia 22134-5027

4. Director, Marine Corps Research Center...2
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, Virginia 22134-5107

5. Director, Studies and Analysis Division...1
   MCCDC, Code C45
   300 Russell Road
   Quantico, Virginia 22134-5130

6. Marine Corps Representative..1
   Naval Postgraduate School
   Code 037, Bldg. 234, HA-220
   699 Dyer Rd.
   Monterey, California 93940

7. Marine Corps Tactical Systems Support Activity..1
   Technical Advisory Branch
   Attn: Maj J.C. Cummiskey
   Box 555171
   Camp Pendleton, CA 92055-5080

8. Chairman, Code CS...1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

9. Dr. Cynthia E. Irvine..2
Computer Science Department Code CS/Ic
Naval Postgraduate School
Monterey, CA 93943-5000

10. Daniel F. Warren...1
Computer Science Department Code CS/Wd
Naval Postgraduate School
Monterey, CA 93943-5000

11. Dr. Blaine Burnham...1
National Security Agency
Research and Development Building
R23
9800 Savage Road
Fort Meade, MD 20755-6000

12. CAPT Dan Galik...1
Space and Naval Warfare Systems Command
PMW 161
Building OT-1, Room 1024
4301 Pacific Highway
San Diego, CA 92110-3127

13. Commander, Naval Security Group Command...1
Naval Security Group Headquarters
9800 Savage Road
Suit 6585
Fort Meade, MD 20755-6585
ATTN: Mr. James Shearer

14. Mr. George Bieber..1
Defense Information Systems Agency
Center for Information Systems Security
5113 Leesburg Pike, Suite 400
Falls Church, VA 22041-3230

15. CDR Chris Perry...1
N643
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington, VA 22202

16. CDR Chris Perry..1
N643
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington, VA 22202

17. Mr. James P. Anderson...1
Box 42
Port Washington, PA 19034

18. Captain Jason Hackerson...1
2541 Babcock Rd.
Vienna, Virginia 22181