Secured Single Signon in a Client/Server Environment

Document Number GG24-4282-00

August 1994

International Technical Support Organization
Poughkeepsie Center

Take Note!

Before using this information and the product it supports, be sure to read the general information under “Special Notices” on page xi.

First Edition (August 1994)

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. H52 Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

Abstract

This document provides an overview of a *Secured Single Signon* solution in a Client/Server environment. This solution is based on the design principals of the *RACF Secured Signon* facility and a end-user verification and authentication facility that is implemented on a smart card (the IBM Personal Security card) that can be read from the workstation or client machine. RACF is not needed to implement the principals of this Secured Single Signon solution. This document describes the various components of this solution. It also provides a high-level overview of the various programs that are part of this solution.

This document is intended to help system programmers, security administrators, network administrators, workstation and client/server application programmers to understand, develop, and implement a Secured Single Signon procedure in a client/server environment.

(112 pages)

Contents

Figures

Special Notices

This publication is intended to help system programmers, security administrators, network administrators, and workstation and client/server application programmers to develop and implement a Secured Single Signon procedure for various client/server applications.

The information in this publication is not intended as the specification of any programming interfaces that are provided by the various software components that are described in this document. See the PUBLICATIONS section of the IBM Programming Announcement for these software components for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms, which are denoted by an asterisk (*) in this publication, are trademarks of the International Business Machines Corporation in the United States and/or other countries:

ACF/VTAM	AIX	AIX/6000
AS/400	AT	CICS
CICS/ESA	ES/3090	ES/9000
IBM	Micro Channel	MVS/ESA
MVS/SP	MVS/XA	NetView
OpenEdition	OS/2	OS/400
Personal Security	POWERserver	POWERstation
PROFS	PS/2	RACF
RISC System/6000	RS/6000	SAA
System/370	System/390	VM/ESA
VTAM		

The following terms, which are denoted by a double asterisk (**) in this publication, are trademarks of other companies:

DCE	Open Software Foundation, Inc.
MS-DOS,	Microsoft Corporation
Microsoft	Microsoft Corporation
Microsoft, Windows	Microsoft Corporation
OSF	Open Software Foundation, Inc.
POSIX	Institute of Electrical and Electronic Engineers
UNIX	Novell, inc

Other trademarks are trademarks of their respective companies.

Preface

This document describes implementation of a Secured Single Signon facility in a client/server environment. This solution assumes the availability of a cryptographic facility on a smart card and a smart card reader on a OS/2, DOS or AIX workstation or client machine The solution is based on the design principals of the Resource Access Control Facility (RACF) Secured Signon function. RACF itself in not needed to implement the principals of this solution in a client/server environment.

The RACF Secured Signon provides an alternative to the password associated with specific user profiles. The alternative is called a PassTicket and allows workstations and client machines in a client/server environment to communicate with the application server without using a password.

The described Secured Single Signon facility improves the security in a network since there are no passwords conveyed in clear text across the network. It also minimizes problems of password management for the end users since the user only has to authenticate (by using a password or PIN) himself to the first authentication server in the network he is logging on to. Once the user is authenticated, he will use PassTickets to sign on to the application servers he is authorized to use.

This Secured Single Signon service concept is based on a the usage of an IBM Personal Security card that will generate the PassTicket for the user that is logging on to the application server. Input for the PassTicket generator is also stored on the Personal Security card. Workstations and client machines that are to be used in this Secured Single Signon approach need to have smart card reader connected to them.

The IBM Personal Security card is a smart card designed as part of the Transaction Security System product family. It is a portable security device, with cryptographic and portable file capabilities. It carries authorization information which controls the user's capabilities, both on the smart card and in the workstation or client machine where it is used. All of the features are configurable by the customer in order to meet a wide variety of needs.

The IBM Personal Security card is available in several different versions, each optimized to meet different customer requirements.

This document is written for MVS system programmers, workstation and client/server application programmers, security administrators, and network designers who are involved in securing the logon procedures to application servers.

How This Document is Organized

The document is organized as follows:

- Chapter 1, "Introduction"

  This chapter provides an overview of the requirements in an client/server environment. This chapter addresses in more detail two security areas: secured signon and single signon.

This chapter also gives on overview of a client/server environment and the position of the Secured Single Signon service offering, related to other solutions, in this environment.

- Chapter 2, “Solution Highlights”

  This chapter provides an overview of the functions and the various components of the Secured Single Signon solution.

- Chapter 3, “RACF Secured Signon”

  This chapter provides an overview of the RACF Secured Signon function. The Secured Single Signon is based on the design principals of this RACF function.

- Chapter 4, “IBM Personal Security Card”

  This chapter provides a detailed description of the IBM Personal Security card. It also describes the implementation of the PassTicket generator on this smart card

- Chapter 5, “Workstation Environment”

  This chapter provides a overview and a high-level description of the hardware and software that is used to design and test the Secured Single Signon solution.

- Chapter 6, “Signon Application”

  This chapter describes the signon application that runs on the workstation. The design principals for this application can be used to design signon applications for other platforms.

- Chapter 7, “Host Software Configuration”

  This chapter provides an overview and a high-level description of the host software that was used to design and test the Secured Single Signon solution.

  It also provides a description on how to enable the RACF Secured Signon facility.

- Chapter 8, “Solution Security Features”

  This chapter provides an overview of the various security features of the Secured Single Signon solution.

- Chapter 9, “What is the Time?”

  This chapter provides an introduction to the way the time is obtained in the various solution components.

- Appendix A, “IBM Cryptographic Facilities Highlights”

  This appendix provides an overview of the currently available IBM cryptographic facilities.

- Appendix B, “Network Security Program”

  This appendix provides an overview of the IBM Network Security Program.

- Appendix C, “Applications of the Personal Security Card”

  This appendix provides an overview of how data blocks on the Personal Security card can be used. It also provides a list of sample applications of the Personal Security card that are suggested by the data block usages as described in this appendix.

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

Resource Access Control Facility (RACF)

- *RACF General Information*, GC23-3723
- *RACF Security Administrator's Guide*, SC23-3726
- *RACF General User's Guide*, SC23-3728
- *RACF Auditor's Guide*, SC23-3727
- *RACF System Programmer's Guide*, SC23-3725
- *RACF Macros and Interfaces*, SC23-3732
- *RACF Command Language Reference*, SC23-3731
- *Enhanced Exploitation of RACF 1.9.2 - Secured Single Signon Using NV/AS*, GG24-4184

IBM Common Cryptographic Architecture

- *IBM CCA: Cryptographic Application Program Interface Reference*, SC40-1675
- *IBM CCA: Cryptographic Application Program Interface Reference - Public Key Algorithm*, SC40-1676

IBM Transaction Security System

- *TSS General Information and Planning Guide*, GA34-2137
- *TSS Concepts and Programming Guide: Volume I, Access Controls and DES Cryptography*, GC31-3937
- *TSS Concepts and Programming Guide: Volume II, Public-key Cryptography*, GC31-2889
- *TSS Programming Reference: Volume I, Access Controls and DES Cryptography*, SC31-2934
- *TSS Programming Reference: Volume II, Public-key Cryptography*, SC31-2888

IBM Integrated Cryptographic Service Facility/MVS

- *IBM ES/9000 ES/3090 ICRF User's Guide*, GA22-7142
- *ICSF/MVS General Information*, GC23-0093
- *ICSF/MVS Administrator's Guide*, SC23-0097
- *ICSF/MVS Application Programmer's Guide*, SC23-0098
- *ICSF/MVS System Programmer's Guide*, SC23-0096
- *ICRF and ICSF/MVS Implementation Guide*, GG24-3802

OpenEdition MVS

- *MVS/ESA: Introducing OpenEdition MVS*, GC23-3010
- *MVS/ESA: Planning: OpenEdition MVS*, SC23-3015
- *MVS/ESA: OpenEdition MVS User's Guide*, SC23-3013
- *OpenEdition MVS for MVS/ESA 5.1 Presentation Guide*, GG24-4095

IBM Personal Communications/3270

- *IBM Personal Communications/3270 Version 3.1 Administrator's Guide*, S84G-0585
- *IBM Personal Communications/3270 Version 3.1 Programmer's Guide for Full-Function DOS Mode and Windows Mode*, S84G-0602
- *IBM Personal Communications/3270 Version 3.1 Programmer's Guide for Entry-Level DOS Mode*, S84G-0601
- *IBM Personal Communications/3270 Version 3.1 Implementation Guide*, GG24-4173

NetView Access Services(NV/AS)

- *NV/AS V2R1 General Information*, GH12-5808
- *NV/AS V2R1 User's Guide*, SH12-5809
- *NV/AS V2R1 Administration*, SH12-5810
- *NV/AS V2R1 Customization*, SH12-5812

Network Security Program(NetSP)

- *Network Security Program Developer's Guide*, SC31-6500
- *Using Network Security Program on AIX, OS/2 and DOS Platforms*, GG24-4149

MVS/ESA

- *MVS/ESA SP V5 Initialization and Tuning Guide*, SC28-1451
- *MVS/ESA SP V5 Initialization and Tuning Reference*, SC28-1452
- *MVS/ESA SP V5 System Commands*, GC28-1442

Acknowledgments

This publication is the result of a residency conducted at the TSS development lab in Charlotte NC and the ITSO in Poughkeepsie N.Y.

The authors of this document are:

Finn Thunbo Christensen	IBM Denmark
Andries Mulder	IBM Netherlands
Antti Numminen	IBM Finland
Harri Levanen	IBM Finland

The support of George Dolan, Todd Arnold and Eric Smith of the TSS Development team in Charlotte NC, Rich Guski of the RACF Development team in Kingston NY, and Bill Ogden of the ITSO was of great help during the design and testing process of the Secured Single Signon facility.

IBM also gratefully acknowledges the contribution of Gunnar Myhre, auditor of Coopers & Lybrand in New York.

Cees Kingma
International Technical Support Organization, Poughkeepsie

Chapter 1. Introduction

The extensive use of open networks and distributed systems pose a serious threat to the security of the end-to-end communications of applications. The increasing numbers of applications and the various security implementations on the different platforms does raise the need for a simplified and secure signon facility for the end user.

1.1 User Requirements

This solution addresses the requirements that were expressed in several GUIDE and SHARE user group meetings. These requirements are:

- Enterprise-wide security.
- Secured Signon; no clear passwords on the network.
- Single Signon; a user should be authenticated by the system once.
- Alternative user authentication.
- Low implementation cost.

This chapter addresses in more detail two security areas: secured logon and single logon. These areas, together or separately, are important for a wide range of installations. In this book, the terms “logon” and “signon” are used interchangeably. The term “Secured Single Signon” is used for a combination of both facilities; secured logon and single logon.

1.1.1 Enterprise-wide Security

The elements that are used to build this Secured Single Signon solution are available or can be installed or implemented on all the different platforms in a client/server environment. The RACF* Secured Signon PassTicket generate algorithm is published, and can be installed in any type of client in the client/server environment.

Note: Vendors that intend to implement the PassTicket technology should contact the IBM Intellectual Property and Licensing Services through the IBM Director of Licensing.

Currently there are several products available in a client/server environment that support the RACF Secured Signon facility. Among these is the IBM* Network Security Program (NetSP). MVS/ESA* host server applications, such as CICS*, IMS, and TSO, and the MVS/ESA session manager NetView Access Services support the PassTicket validation if the RACF Secured Signon function is enabled on the MVS/ESA environment.

The cryptographic facility that is used in this solution, or similar facilities from other vendors, is also available on every type of client or server machine.

1.1.2 Secured Signon

Conventional systems use passwords to authenticate a user. Such systems may cause security exposures since these passwords are usually conveyed in the clear across the network, and could therefore be intercepted and used illicitly. Equipment capable of tracing data that flows across both local and wide area networks is becoming increasingly prevalent. Also, passwords often appear in the clear in traces and dumps.

Since users often select trivial passwords, they are easily guessed and might be an eye-catcher in storage dumps or line trace outputs. Filter criteria in security managers can detect some of these trivial passwords, but not all of them. Installation-assigned passwords present many other problems; the most important is that users invariably write these down, often next to their terminals, and installation-assigned passwords are usually also not protected against re-use once they are intercepted.

In other words, any system that sends clear passwords over the network (either LAN or WAN) should be regarded as insecure, and a candidate for a secured logon facility.

RACF Secured Signon function provides an alternative to the password associated with specific user profiles. This alternative password, or PassTicket, allows workstations and client machines in a client/server environment to communicate with server applications without the clear-text password.

1.1.3 Single Signon

With the increasing use of distributed systems, a user may need to access a number of systems, either explicitly, through manually logging on to these systems, or implicitly, where an application running on a client workstation will interact with a number of server applications.

In either case, password management is a problem. The user is required to maintain passwords for all systems used, ideally keeping them all unique, difficult to guess, and without writing them down, or requiring the systems to perform password maintenance, synchronization, and similar functions.

One common approach to a single signon is to use “recorded logons.” The user records a particular logon sequence, including the password. This recorded logon is played back whenever the user wants to log on to an application. This solves the problem of remembering the password for that application, but introduces new problems, namely:

- The recording must be changed every time the password changes.
- The recorded password, in a local file, is a security exposure.
- The password is still sent in clear across the network.

Ideally, a user should sign on and be authenticated by the system once. The authenticating system should then warrant the user′s authenticity to other systems or applications the user requests access to, without transmission or request for ID and password.

1.1.4 Alternative User Authentication

The Secured Single Signon solution removes the need to send passwords across the network and enables a trusted function other than the server to authenticate a server application user ID. Once authenticated, the user can be authorized to use PassTickets to sign on to various applications in different application servers. The use of PassTickets is not limited to RACF protected host applications. Every application server that is capable of installing the PassTicket generation algorithm can be part of this secured logon environment.

The inclusion of a cryptographic facility enables the user to select a different end-user verification and authentication procedure. By using the IBM Personal Security* card, the user verification and authentication can now be based on two items:

- Something the user must supply - his Personal Security card
- Something the user must know - his Personal Identification Number or Personal Security card password

1.1.5 Low Implementation Cost

The solution as described in this book can be made available for a very reasonable price. Depending on function requirements, the price should be less than $200 USD per station including a Personal Security card, a card acceptance device (reader), and client machine software.

1.2 Client/Server Environment

A sample configuration of a client/server environment can consist of the following components:

- An MVS/ESA host with several business applications that run under, for example, CICS, TSO, or IMS.

  This host can act as an *authentication server* and as an *application server* for various components in the environment.

- A Local Area Network (LAN) with several client machines, an authentication server, and a application server connected to it. Servers and client workstations on this LAN can communicate with the host through various types of network connection.

- A programmable workstation (PWS) with, for example, a DOS, OS/2*, or AIX* control program. This PWS will perform a 3270 emulation session with the host application server through a network connection.

  Attached to this PWS is a smart card reader that can communicate with an IBM smart card. This smart card, with a cryptographic processor on it, will act as the authentication server for this workstation.

- A locally or remotely attached 3270 terminal.

An example of this configuration is shown in Figure 1.

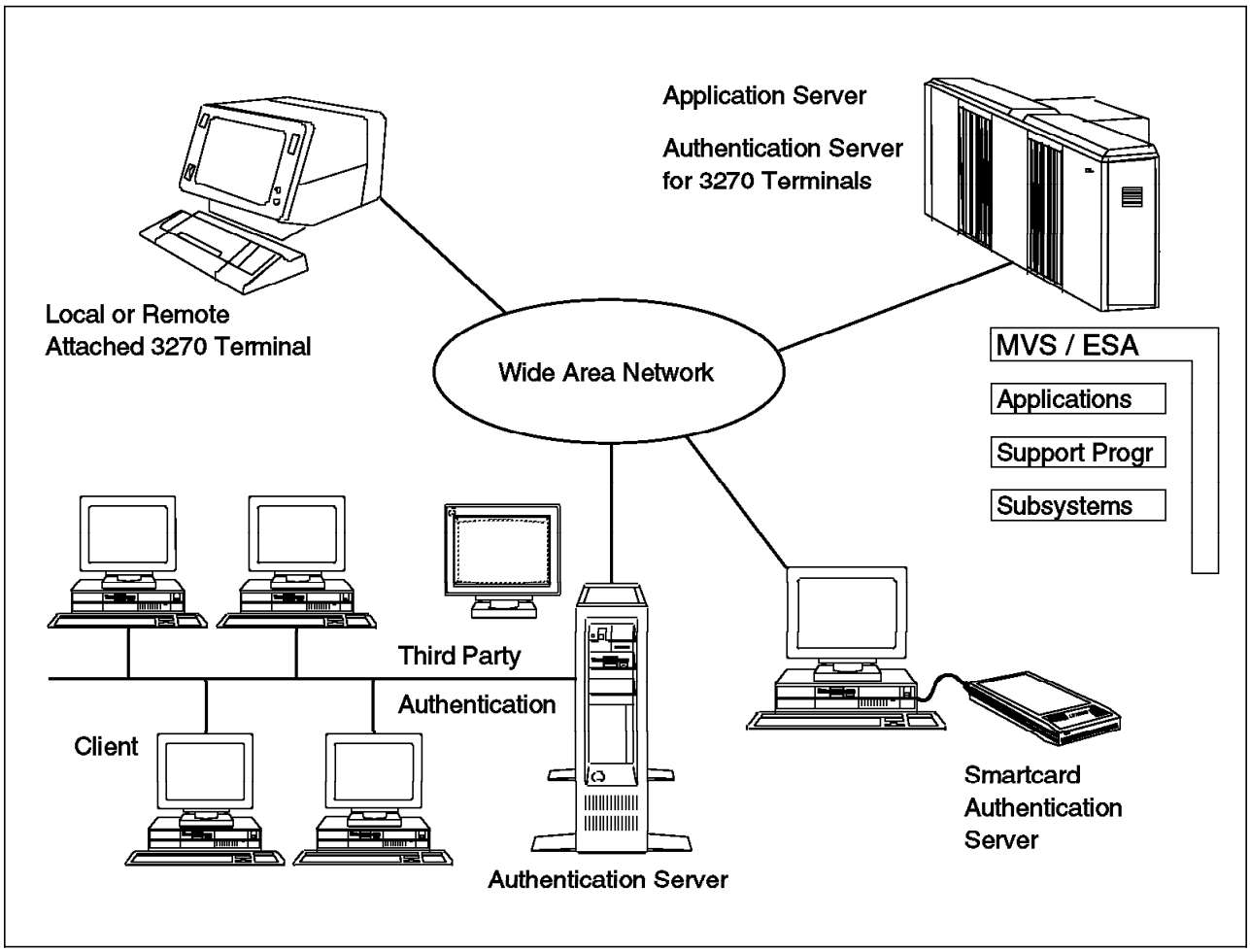

Figure 1. Example of Client/Server Environment

1.2.1 User Authentication in a Client/Server Environment

A user at any workstation or client machine in this environment will authenticate himself to any of the authentication servers by supplying one of the following combinations:

- A user ID and a password. This is the only password that the user is required to remember. In most cases, this password is *not* sent across the network as clear text.
- A smart card and his Personal Identification Number. The user does not need to know the user IDs and passwords for the server applications he is using.

1.2.2 Local or Remote Attached 3270

A local or remote attached 3270 terminal still has a password in clear text on the communication link between the terminal and the first authentication server on the host. Once the user is authenticated in, for example, the session manager NetView Access Services (NV/AS) on the host, the user can be authorized to use PassTickets. This provides the user with a single signon facility since he does not have to know the passwords for the applications he is accessing through the session manager.

This implementation is described in *Enhanced Exploitation of RACF 1.9.2 - Secured Single Signon Using NV/AS*.

1.2.3 Workstation with 3270 Emulation

If the user is using a programmable workstation that is performing a 3270 emulation session with the host application server, Secured Single Signon can be implemented by using the Secured Single Signon solution, as described in Chapter 2, “Solution Highlights” on page 7.

1.2.4 Stand-alone Client Machine

If the user is using a AIX system that is performing a session with a server application, Secured Single Signon can be implemented by using the Secured Single Signon solution, as described in Chapter 2, “Solution Highlights” on page 7.

1.2.5 LAN Connected Client Machine

When the user is using a programmable workstation that is connected to a LAN, there is no need to have clear text passwords on the network. Client workstation code, which is part of the Network Security Program (NetSP), allows the user to authenticate himself to the authentication server that is part of NetSP, without sending a clear text password over the LAN. Once the user is authenticated, the NetSP authentication server generates PassTickets for the applications the user is authorized to access.

See Appendix B, “Network Security Program” on page 95 for a short introduction to NetSP.

1.2.6 Passing PassTickets

Since PassTickets are one-time-only and nonreusable passwords, they are protected against replay. The PassTicket is passed to the host application in the password field in a normal log-on screen. The application passes the PassTicket to RACF for validation instead of the usual password. Once the PassTicket reaches RACF, it performs the following processing: if the PassTicket supplied does not match the conventional password stored for the user ID, RACF performs PassTicket validation.

Chapter 3, “RACF Secured Signon” on page 15 provides a description of the RACF Secured Signon function. *RACF Macros and Interfaces* provides a complete description of the PassTicket generate and validate algorithm.

1.3 Host Session Manager

Host session managers, such as NetView Access Services, simplify the task of gaining access to host applications and enable the end-user to work with several applications from a single workstation or client machine at the same time.

An application can be a MVS/ESA subsystem (for example, IMS/VS or CICS/ESA*), an application (for example, a E-Mail system like PROFS*), or a transaction within a subsystem.

The usage of a host session manager also simplifies the administration and improves the security in a client/server environment. Only the host session manager needs to be defined in the authentication server. Security sensitive

information, such as the access control to the various host applications and input to the PassTicket algorithm, can be kept in the secure host environment (for example, in the RACF database).

See *Enhanced Exploitation of RACF 1.9.2 - Secured Single Signon Using NV/AS* for a comprehensive overview of such an implementation.

The authentication server in Network Security Program provides the same type of facility. Information about the server applications can be stored in the *principal database*. This principal database can be loaded with an extract from the host RACF database.

1.4 What is a Smart Card?

Smart cards go by several different names, including *IC Card* and *Chip Card*. A smart card is identical in appearance to a standard plastic credit card. The difference is that the smart card contains an embedded microprocessor, capable of storing data and performing some kind of intelligent processing. There is a broad range of smart card capabilities, with differences in processing power, memory type, and memory size. One significant difference is between cards using EPROM memory and those that use EEPROM. EPROM can be programmed only once, then the contents cannot be changed. EEPROM, on the other hand, can be reprogrammed. Both are nonvolatile, allowing the cards to remember data that is stored in their memories.

1.5 What is the Personal Security Card?

The Personal Security card (PSC) is IBM′s first entry in the smart card field. It is at the high end of smart card technology and capability. The card contains an 8-bit microprocessor, with on-chip RAM, ROM, and EEPROM memory. The architecture of the Personal Security card, with the rest of the Transaction Security System (TSS), evolved from experience gained through several joint studies and pilot projects with customers.

Figure 2 shows a IBM Personal Security card.

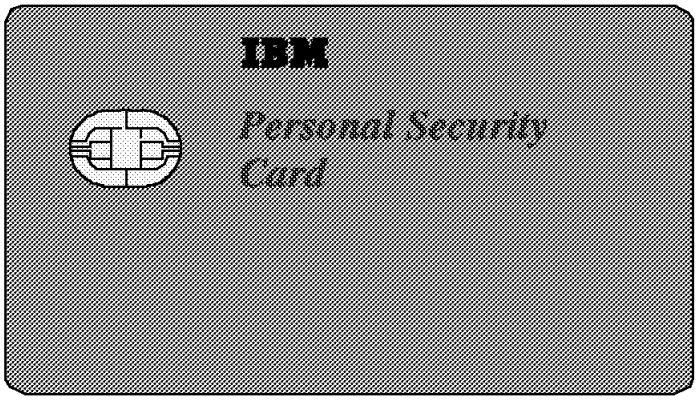

Figure 2. IBM Personal Security Card

Chapter 2. Solution Highlights

This chapter provides an overview of the Secured Single Signon solution. The solution is based on the design principals of the RACF Secured Signon function and a IBM cryptographic facility; the IBM Personal Security card. This high-level overview describes the various components that are involved in this solution. This chapter also provides a description of the processes in this solution.

The proposed solution is based on the following RACF Secured Signon characteristics:

- The RACF Secured Signon PassTicket algorithm is published and is portable to different workstation and client/server platforms, such as DOS, OS/2, AIX, OS/400*, MVS/ESA, and VM.
- The input for the PassTicket algorithm can be made available in all the above-mentioned platforms.
- Using a smart card to store the secret application keys ensures maximum security for these keys.

Note: The secret application key must be protected against unauthorized viewing. Once this key is exposed, every user with access to a PassTicket generator facility can generate PassTickets for every other user that is authorized for that particular application.

2.1 Solution Component Types

The Secured Single Signon solution consists of the following types of components:

- Standard product functions, such as the RACF Secured Signon that needs to be tailored for this solution. For example, a RACF class must be activated and profiles for that class must be defined at the host to enable the RACF Secured Signon function.

  If RACF is not available on components in the client/server environment, the PassTicket generate and validate algorithm needs to be implemented on these components.
- Nonstandard product functions, such as the special version of the IBM Personal Security card, that need to be ordered as part of this Secured Single Signon solution.
- User written applications, such as the signon application that needs to be executed on the workstation or client machine to call the PassTicket generating routine on the smart card.

Sample programs are available as part of this Secured Single Signon solution.

2.2 Solution Benefits

This solution introduces benefits in the following areas; ease-of-use for the end user and the security administrator, enhanced network security, and support in a heterogeneous environment.

2.2.1 Ease-of-Use for the End User

The end user does not have to change or remember his password, user ID, and application name for the application he wishes to logon to. All the user needs is the Personal Security card, which can also be used for several other purposes, like an ID card, cash card, and so forth. He also needs to know his PIN. It could not be simpler for the end user.

The method for logging on to all server applications is uniform. No matter which application platform the user selects to access, it will look and feel the same. Even though it is done differently under the covers through a *logon script file* for each application, the end user sees the same interface for all applications.

2.2.2 Ease-of-Use for the Security Administrator

Password reset and ID resume will not be a needed service since passwords are not used, and forgetting and guessing passwords will not occur. No IDs will be revoked, so resetting IDs will not have to be done.

Creating a Personal Security card for users is a straight forward, one time activity per user and done as part of the normal user registration process.

Since passwords are not used, there is no need for password synchronization between multiple RACF systems. This a problem for several large installations with multiple mainframes.

2.2.3 Network Security

There are no ″useable″ passwords transmitted over the network in clear text. Linetapping will not disclose information useable for any system access since a one-time password is generated at each application signon.

Control of user access to applications can be done without any special definitions in the server application. Application access control can be done at both ends of the session.

2.2.4 Support in a Heterogeneous Environment

With PassTicket validation made available on several other platforms, it could be possible to implement a true enterprise Secured Single Signon solution. For each platform where user IDs and passwords are used to authenticate a user, a PassTicket could be substituted for the password.

2.3 Solution Design Objectives

By using the IBM Personal Security card, it is possible to achieve a very secure implementation of a PassTicket generation facility.

The global design for this Secured Single Signon solution is based on the following:

- Prevention of a malicious user from modifying the PassTicket code or from intercepting an intermediate value of the PassTicket algorithm. The PassTicket algorithm and the secret application key that is needed to generate a PassTicket need to be stored together on the Personal Security card.

- The secret application key cannot be used for any purpose other than the PassTicket generation on the smart card. The key cannot be used in any other function outside the smart card.
- A user profile is stored on the card. Only the user who knows the right Personal Identification Number or password can operate the PassTicket generator on this Personal Security card.
- A list of applications the user is authorized to access and the user ID for those applications is stored on the Personal Security card.
- Although the user has access to some of the information on the smart card, after he is authorized with a valid PIN or password, he cannot change the relationship between the application he is using and the user ID that is used.
- The design should allow storage of multiple application keys on the smart card. This includes spare keys that can be activated in case the active key is exposed.

2.4 Security Features

This implementation applies the principle that once the user is authenticated by a trusted party, he can generate a PassTicket for each application he is authorized to.

A secured logon to the various applications is achieved by sending a PassTicket instead of a clear-text password in the logon sequence.

Since the PassTicket generation algorithm is loaded on the Personal Security card, the secret application key is never available, in the clear, outside the Personal Security card. This prevents the key from being exposed for unauthorized viewing.

The list of authorized applications contains a pointer to the secret application keys that are needed to generate a PassTicket for these applications. These keys are stored in the data key table on the Personal Security card.

2.5 Signon Application Flow

After the implementation of this Secured Single Signon solution, the flow for the signon application on the workstation or client machine can be as follows:

1. User inserts the Personal Security card in the card reader.
2. User types his PIN or password on the keypad of the smart card reader or on the workstation or client machine keyboard.
3. After successful authentication, a list of applications the user is authorized to use is presented by the signon application.
4. User selects the host applications of his choice.
5. The signon application is replaying a logon script file for the requested application. This logon script file contains “ampersand variable” for the user ID (&UID) and the password (&PT) that will be replaced by real values (user ID and PassTicket) during this signon.
6. At the detection of an ampersand variable, the signon application calls the PassTicket generator routine on the smart card.

7. The smart card produces a PassTicket for the end user that is defined as the requesting user for the requested application.
8. The PassTicket generator on the smart card returns the user ID, the application name, and the PassTicket to the signon application.
9. The user ID and the PassTicket are forwarded through the High-Level Language Application Program Interface (EHLLAPI) of the 3270 emulator program to the signon panel of the requested application.

Refer to Figure 3 for a overview of the flow in the signon application.

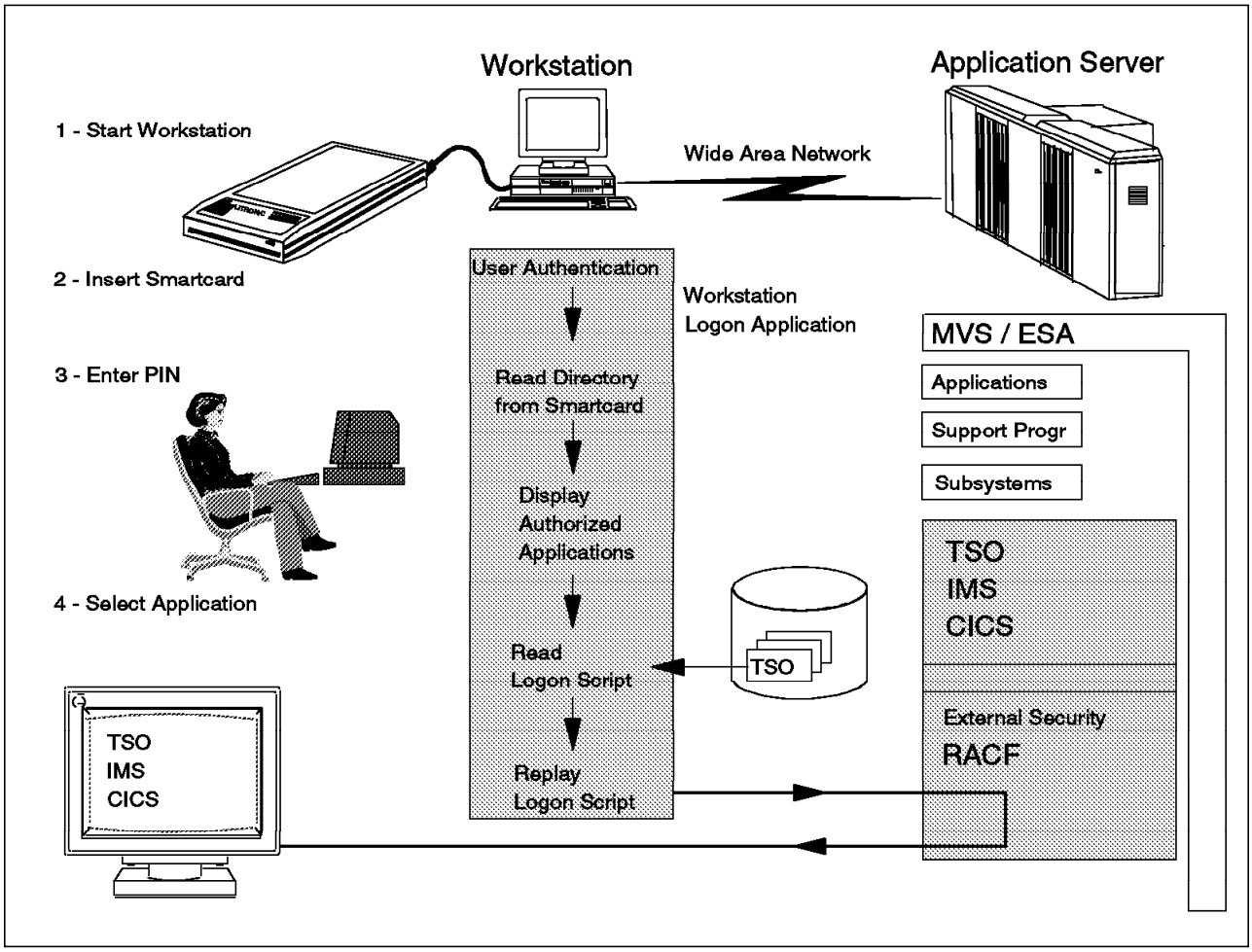

Figure 3. Signon Application Flow

2.6 Solution Elements

The Secured Single Signon solution consists of the following elements:

- Standard available product functions that needs to be tailored for this solution. Functions in products such as:
  - Workstation and client machine operating systems, such as OS/2, DOS and IAX, 3270 emulators or CM/2, client application software.
  - Host operating systems, such as MVS/ESA or VM, sub-systems such as RACF, ACF/VTAM*, NV/AS, TSO/E, IMS and CICS/ESA, and server application software.

  - Network connections, such as CM/2 or PC/3270.
- Nonstandard product functions, such as:
  - Special version of the IBM Personal Security card.
  - Special version of the *Workstation Security Service Program*.
- Special designed applications that run on the host and on the workstation or client machine.
  - Host signon application on the workstation or client machine.
  - Host application to initialize the smart cards.

2.6.1 Server Software

Once the RACF Secured Signon function is enabled and RACF profiles are defined, there are no further modifications needed on the host software configuration. RACF Secured Signon works with existing protocols in the MVS/ESA subsystems such as CICS, IMS, TSO and NV/AS.

If RACF is not available on the server, the PassTicket generate and validate algorithm has to be implemented on the server.

Some updates might be necessary in the audit reports to reflect the use of PassTickets.

2.6.2 Workstation or Client Machine Software

The signon application that is used in this project and that is described in more detail in this document is designed for a DOS-based 3270 emulator. The signon application can run in PC-DOS or MS-DOS**, or in a OS/2 2.1 DOS “box.” The design principles and most of the sample code can be used in any other operating system in a workstation or client machine.

The solution also requires a high-level language programming interface to the 3270 emulator program. In this design, the EHLLAPI interface is used that is associated with the IBM 3270 emulator.

A new TSS Security Application Program Interface (SAPI) call is needed for this solution. This SAPI call will execute the PassTicket generate function on the smart card.

There is special device driver needed for the Argus/200 smart card acceptance device.

2.6.3 Cryptographic Facilities

The PassTicket generate algorithm is installed on a special version IBM Personal Security card; the so called Secured Single Signon Card (SSS Card). This SSS Card can also be ordered as a separate item outside the solution package.

Currently there are several smart card readers available. The design for this solution is based on the Argus/200 smart card acceptance device and on the IBM 4754 Security Interface Unit. For an overview of the available smart card readers, refer to: Appendix A, “IBM Cryptographic Facilities Highlights” on page 79.

2.6.4 Signon Application on the Workstation

This application is written in C language and will execute under DOS. It will perform the automated Secured Single Signon facility. The design of this application and the sample code can be used on other operating systems on workstations and client machines.

2.6.5 Smart Card Initialization Program

Smart card initialization consists of several steps:

- Microcode load, which is done by the manufacturing process of the special version of the IBM Personal Security card. This initialization program will load the PassTicket generation algorithm onto the card.

  Note: This step can only be performed by IBM manufacturing and is not part of the initialization task that needs to be performed by the customer.

- TSS component initialization, where things like access control information is loaded onto the smart cards. A special setup is provided to use the standard TSS Hardware Initialization and Key Management (HIKM) utility to load the smart card with the correct information for this solution.
- Application initialization, which loads the user and application dependent information, such as user ID and secret application key, onto the smart card. A special program is designed to load this information onto the card.
- A host application to retrieve the secret application key from the RACF database and send it across to the workstation of the security administrator that is responsible for the smart card initialization.

2.7 Security Considerations

By implementing the Secured Single Signon facility, the identification and authentication of the user ID and the access control to the host application is moved from the target host application to the smart card.

The security controls, such as

- Identification and authentication
- Access control
- Confidentiality
- Integrity

on cryptographic facilities like the smart card are comparable to, and in some cases exceed, the controls in a MVS/ESA - RACF environment.

Refer to 4.4, “Authorization - Overview” on page 26 in Chapter 4, “IBM Personal Security Card” for an introduction to the various security controls that are available on the IBM smart card.

Refer also to Chapter 8, “Solution Security Features” on page 73 for an overview of protection against attacks on the Secured Single Signon facility on the smart card.

2.8 Audit Requirements

On the host server systems that use the MVS/ESA-RACF implementation, the PassTicket validation logic produces a System Management Facility (SMF) record to indicate a successful signon with a PassTicket. Two *Event Qualifiers* are available in SMF record Type 80 Event 1. The meaning of these Event Qualifiers is as follows:

Qualifier	Meaning
32	Successful initiation using PassTicket
33	Attempted replay of PassTicket

The audit record that is produced to indicate an *unsuccessful* signon attempt (even when a PassTicket is used) is unchanged. If the PassTicket fails, the logic cannot detect if this was a wrong password or an invalid PassTicket.

The RACF Report Writer or the SMF Data Unload Utility can be used to select these records from the SMF database and list them in regular reports.

2.9 Ordering the Solution

This solution described in this book has been implemented as prototype code for the personal computer and Personal Security card. A product solution consists of Personal Security cards that include the PassTicket generation capability, a low-cost card acceptance device (reader), client machine software, a card initialization prototype, and RACF extract support.

Transaction Security System development in the Branch Delivery Systems organization in IBM at Charlotte, NC, USA, should be contacted through your local IBM representative to negotiate a customized solution to meet your requirements.

Chapter 3. RACF Secured Signon

The PassTicket is, in most environments, a *one-time-only* and *nonreusable* password that is generated by a requesting product or function. It is an alternative to the (RACF) password and removes the exposure of sending passwords across the network in the clear.

3.1 Generating a PassTicket

A product or function that generates a PassTicket must use the RACF PassTicket generator algorithm. This algorithm requires specific information as input data and produces a PassTicket that substitutes for a specific end-user's RACF password. RACF uses the PassTicket to authenticate the end user for a specific application executing on a specific MVS or VM system that uses RACF for identification and authentication.

Note: The PassTicket algorithm can also be implemented on systems where RACF is not available. This document describes implementations, both where RACF is and is *not* available.

There are two ways to generate a PassTicket using the algorithm:

1. If the function using the RACF Secured Signon capabilities is running on an MVS or VM system, you can use the RACF Secured Signon callable service to generate the PassTicket. The algorithm is already incorporated into the service and allows RACF to generate a PassTicket on the host:

   - On MVS, an authorized program, such as one authorized by the *Authorized Program Facility* (APF) and running in MVS *System Key Zero* and *Supervisor State*, can use the callable service to generate PassTickets.
   - On VM, only programs written for RACF for VM and operating within the RACF VM service machine can use the callable service.

   See Section 7.7.7, "Using the Callable Service to Generate a PassTicket" on page 71 for more information on the callable service. Refer to *Enhanced Exploitation of RACF 1.9.2 - Secured Single Signon Using NV/AS* for a sample program using this callable service.

2. If the function that generates the PassTicket is not running on a MVS or VM system, you can create a program that incorporates the algorithm.

   For the solution that is described in this book, the PassTicket algorithm is implemented on the IBM Personal Security card. Some of the inputs are stored, and protected by using cryptographic techniques, on the smart card. Other inputs, such as the time parameter, are included in the call to the smart card that produces the PassTicket.

3.2 PassTicket Generator Algorithm Input Data

To successfully authenticate an end user that is using a PassTicket, the following four inputs should be used in the PassTicket generate algorithm.

- The user ID for the target application.
- The secret application key.

- The APPLID of the target application.
- Time and date information. To limit the usability of a PassTicket to a number of seconds, the PassTicket generator needs a time stamp.

  The input data must be a 4-byte binary number that shows how many seconds have elapsed since January 1, 1970, at 00:00 Greenwich Mean Time (GMT).

  In an MVS or VM environment, the algorithm gets this time stamp from the CPU clock. In workstation and client machine environments, the PassTicket generate algorithm will get the input from the local (PC or RS/6000*) clock. To check the GMT in the participating environments, refer to: Chapter 9, “What is the Time?” on page 77.

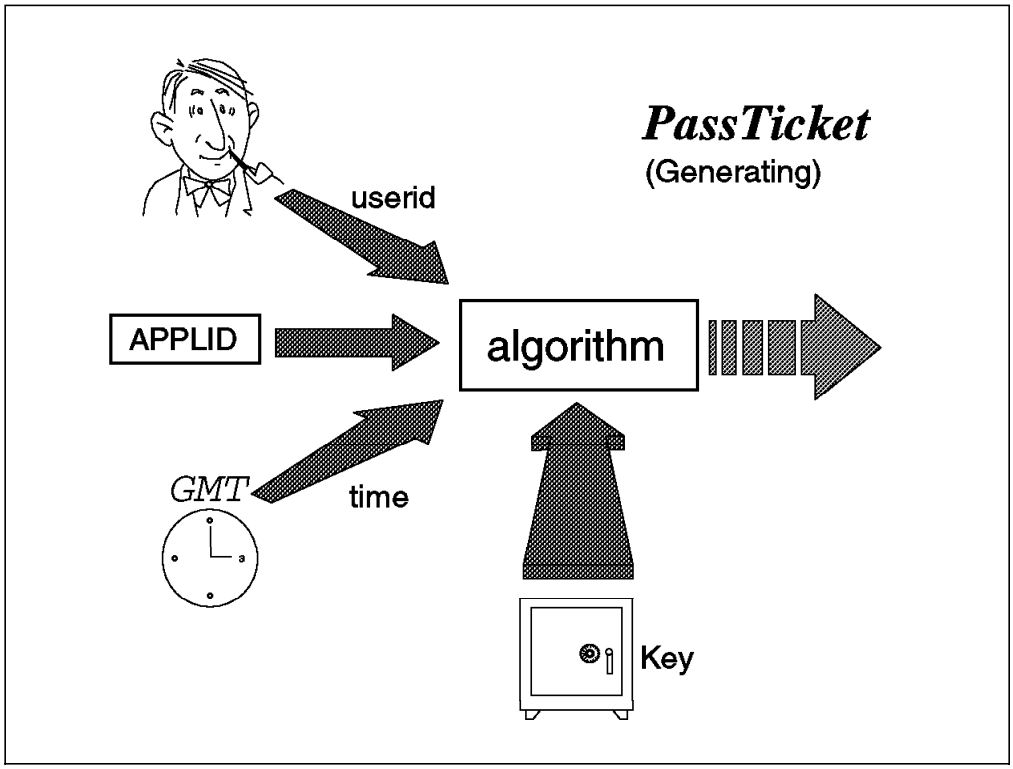

Figure 4. PassTicket Inputs

3.3 PassTicket Algorithm

The PassTicket generator algorithm uses the input information to create a PassTicket. The information passes through the algorithm, which uses a cryptographic technique, to ensure that each PassTicket is unpredictable.

The PassTicket is an 8-character alphanumeric string that can contain the characters A through Z and 0 through 9.

For a complete description of the PassTicket algorithm, refer to *Resource Access Control Facility: Macros and Interfaces*.

3.4 Protecting the Secret Application Keys

To prevent unauthorized usage of the secret application key, make sure that this key cannot be retrieved by an end user. Once the end user knows the key for a particular application, he can generate a valid PassTicket for any user ID for that application since he can find or generate the other necessary input parameters for the PassTicket algorithm. The user then can logon to that application and use another user's authorizations for the application.

In a RACF environment the key can be stored in the RACF database. RACF either masks or encrypts each key. For more information, see 7.7.5, "Protecting the RACF Secured Signon Application Keys" on page 70.

In a workstation and client machine environment, it is difficult to store the key in a protected way in the DOS, OS/2, or AIX file system without using a cryptographic facility. In the solution described in this book, we store the secret key on the smart card in cryptographically protected storage. Only the PassTicket generator, which is also located on the smart card, will get access to the secret application key.

3.5 User Authentication

The RACF Secured Signon function removes the need to send passwords across the network and allows you to remove the user authentication part of the signon from the server to another product or function running in the workstation or the client machine.

If you allow a function or product on the workstation or client machine to authenticate users, that function or product must assume the responsibility of ensuring the security of the user authentication checking function. It should not be possible for the end user to change the relationship between the application he is authorized to use and the user ID he should use for that application.

In the solution that is described in this book, the relationship between the application, the user ID, and secret application key is stored in protected storage on the smart card. Only the PassTicket generator algorithm that is stored on the smart card has access to these, cryptographically protected, inputs for the PassTicket algorithm.

3.6 Enabling the RACF Secured Signon Function

This document describes various implementations of the RACF Secured Signon function on different client/server platforms. Each implementation has its own specific enabling features. The next sections describe the various enabling features.

3.6.1 In a Host RACF Environment

In order to enable the RACF environment to validate PassTickets, the following steps should be taken:

1. Activate the RACF class: PTKTDATA. The PTKTDATA class is the class in which all profiles that contain PassTicket information are defined.
2. Define profiles in the PTKTDATA class. For each application that users are allowed to access with a PassTicket, you must create a profile in the PTKTDATA class. This profile associates a secret Secured Signon application key with a particular application on a particular host system.

Refer to 7.7, “RACF Secured Signon” on page 67 for more information on how to activate the RACF Secured Signon function on the host environment and how to define RACF profiles.

3.6.2 In a Workstation or Client Machine

The PassTicket algorithm can be implemented on a smart card together with the input for this algorithm. Cryptographic techniques should be used to secure the information on the smart card as well as the PassTicket algorithm. The cryptographic facilities on the smart card should also be used to identify and authenticate the end user.

The smart card can be read on every workstation or client machine that has a smart card reader or smart card acceptance device attached to it. A special *signon application* is used to call the PassTicket generate algorithm.

Once this smart card is initialized and loaded with the specific user information, the user can signon to (host) applications without the need of sending clear text passwords across the network. A card initialization program accepts an extract from the host security database as input to load the various data blocks on the smart card.

3.6.3 In a Local Area Network

The Network Security Program(NetSP) provides *third-party* authentication of users and applications between multiple operating systems and multiple transfer protocols. NetSP provides convenience for users since they can sign on once a day and be authenticated as a user of all the secured applications they are allowed to access.

The PassTicket algorithm is implemented on the authentication server of the Network Security Program. Once the user is identified and authenticated by the trusted third-party security program, the user uses PassTickets to signon to various host applications.

Information about the applications and about the end user is stored in the *principal database* in the authentication server of Network Security Program. Input to the PassTicket algorithm, including the secret application key, is also stored in this principal database. The database can be loaded by manual entry through panels or by using a NetSP utility that uses an extract from the host security database as input.

Chapter 4. IBM Personal Security Card

The IBM Personal Security card is a smart card designed as part of the Transaction Security System family of products. It is a portable security device, with cryptographic and portable file capabilities. It carries authorization information which controls the user's capabilities, both on the smart card and in the workstation where it is used. All of the features are configurable by the user in order to meet a wide variety of needs.

The Personal Security card is available in several different versions, each optimized to meet different requirements. For example, the following versions address different needs:

Standard card Complete set of functions, including all DES cryptographic functions.

Commercial DES card Special version for export. The DES Encipher and Decipher functions are removed, but Message Authentication Code and all other cryptographic functions remain.

High capacity card Special version optimized for maximum available memory space. Some little-used functions are deleted to free up EEPROM memory for use by the customer.

CDMF card Replaces DES encryption with CDMF, the Commercial Data Masking Facility. This is a fully exportable cryptographic algorithm with 40-bit key strength, which uses the same data and key formats as DES.

SSS Card The Secured Single Signon Card is a special version with the PassTicket generator algorithm on the card. Some little-used functions are deleted to free up EEPROM memory for use by this solution.

4.1 An Overview of the Personal Security Card

The smart card contains an advanced 8-bit microprocessor with 256 bytes of RAM, 10K bytes of ROM, and 8K bytes of EEPROM memory on the same chip. The face of the card has eight metal contacts, of which five are used:

- Power and Ground
- Reset
- Clock
- I/O

The card communicates with the reader using asynchronous serial communications over the single I/O line, at a speed of up to 96,000 bits per second.

Note: This assumes a clock frequency of 9.216 MHz. The 4754 reader uses half this frequency, so the baud rate is reduced to 48,000 bits per second.

The Personal Security card features can be grouped into three main areas: cryptography, user authentication and authorization, and portable data storage. These areas will be discussed in detail in the sections that follow.

4.1.1 Personal Security Card Security Features

The Personal Security card has a number of security advantages over most competitive cards.

- It has DES cryptographic functions (approximately 2,000 bytes/second internal encryption rate).
- It supports the Common Cryptographic Architecture, which makes it compatible with the entire TSS family of security products.
- It can establish secure cryptographic sessions with other devices, allowing secure communications. (See 4.3, “Secure Sessions” on page 25)
- It has extensive authorization control features.
- It has extensive data access control features.
- It uses a single chip for the processor and all memory. Some smart cards use multiple chips, with exposed interfaces between the processor and the memory. These exposed interfaces can be probed, revealing the contents of the memory.

The card can be enhanced using its IML features to download microcode extensions or patches. IML includes features to prevent security exposures when new code is added to the card.

4.1.2 EEPROM Memory Layout

The smart card contains 8K bytes (64K bits) of read/write, non-volatile EEPROM memory. This memory is shared among several functions.

- There are fixed data areas which are used by the internal microcode. These areas include such things as the DES Master Key, the User Profiles, configuration information, and the card serial number and part numbers.
- Any additional microcode downloaded to the smart card will occupy part of the EEPROM.
- There is an error log area.
- There are two DES key tables, one for Data Keys (KD table), and one for Key Encrypting Keys (KEK table).
- The remainder of the EEPROM space is available for user defined *Data Blocks*, which are described in detail later.

Figure 5 summarizes the layout and contents of the smart card EEPROM.

4.1.3 EEPROM Configurability

Each customer will have a unique application for the smart card, and each will need different numbers of data keys, key encrypting keys, and so forth. To optimize the use each customer can get from the EEPROM, he has the capability to configure the memory layout himself. When the customer receives the cards, he has the option of loading *Global Configuration Data* to the card, which will customize the EEPROM configuration to meet his needs.

The customer can choose the sizes of the error log, the data key table, and the key encrypting key table. These, in turn, determine the amount of space remaining for customer defined data blocks, where customer specific data can be stored. By only allocating space for the number of keys he will use, the space available for data blocks is maximized.

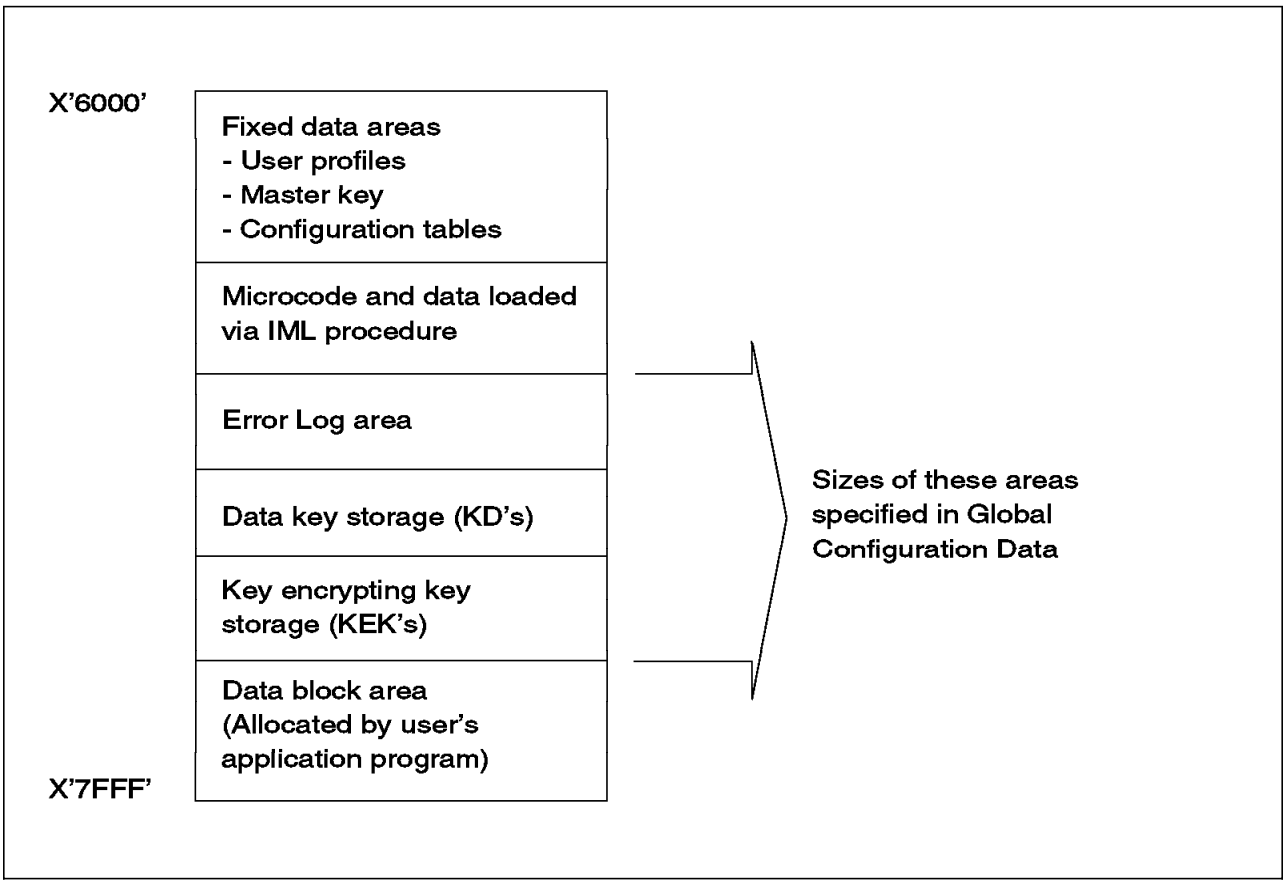

Figure 5. EEPROM Layout

The number of data keys can be anywhere from 0 through 255. The number of key encrypting keys can also range from 0 through 255. The size of the error log can be from 0 through 2040 bytes. The maximum sizes of all three areas will not fit in the EEPROM, but the customer can choose whatever combination is most appropriate for his application.

Note: The card initialization procedure that is part of the Secured Single Signon solution consists of several steps. It configures the various EEPROM areas, loads the relevant information on the card, such as the user ID, application name, and secret application key, and protects the various fields. It also loads the various user profiles on the smart card.

4.2 Command Set

The smart card can execute a large variety of commands. Command requests are transmitted to the card, the requested operation is performed, and the response message is returned to the requesting device.

The commands in the smart card command set can be grouped into several categories with related functions.

Notes:

1. The PassTicket generate algorithm is added to the SSS Card. The PassTicket generate service is available on the SSS Card as a normal executable command.
2. Not all the commands are available on the SSS Card. Some little-used, functions are deleted to free up EEPROM memory.

4.2.1 Key Management Commands

The smart card uses Key Management commands to handle the keys it needs for cryptographic operations. Some of the commands in this category are:

- Import and export data keys and key encrypting keys. These functions bring keys into the smart card's internal key tables, or send keys from the tables back out of the card. When keys are transported to or from the card, they are encrypted using Key Encrypting Keys.
- Load the smart card master key. The master key is loaded in two or more parts, which are combined to form the final key. In this way, two or more people have to cooperate to produce the master key, and none of them individually possesses the key.
- Load clear key encrypting keys. Like the master key, key encrypting keys can be loaded in clear text using multiple key parts from separate people. Note that key encrypting keys can also be loaded in encrypted form using the key import function mentioned above.
- Re-encipher to master key, re-encipher from master key, and re-encipher to new master key. Keys can be enciphered under the smart card master key and stored outside the card. These keys can subsequently be translated to encipherment under a key encrypting key, and loaded into the card or exported to another device. Keys enciphered under the master key can be updated to a new master key if one is loaded to the card.

4.2.2 Cryptographic Commands

These commands use the card's cryptographic capabilities to process customer data or to maintain secure communication among devices. The cryptographic commands include:

- Encipher or decipher data in DES Cipher Block Chaining (CBC) mode
- Generate or verify a MAC (Message Authentication Code)
- Establish a randomly derived session key to secure communications between the smart card and another device

4.2.3 Configuration and Initialization Commands

Several commands are used to configure the card (see 4.1.3, "EEPROM Configurability" on page 20) and to initialize it with data for the user's application. Many tables and configuration data areas are available, as described in later sections. Some of the commands used for these functions include:

- The *Load Tables* command is used to load *user profile* data, card identification fields, and other tables.

- The *Load Configuration Data* command is used to load the *global configuration data* described under 4.1.3, “EEPROM Configurability,” as well as other configuration data fields that will be described later.
- The *Change PIN* command allows the user to change the PIN he uses to identify himself to the smart card.

4.2.4 User Verification

There are two ways a user can identify himself to the Personal Security card and the other TSS components.

1. He can use a PIN (Personal Identification Number), much as PINs are used with ATM cards at banks today. The PIN is entered on the Security Interface Unit keypad, which encrypts the number under a session key and sends it to the smart card. The card decrypts the PIN value and compares it to one of the PINs it has stored internally.

   The PIN can be any length, up to 16 digits.

   Note: The user should enter his PIN on the workstation or client machine keyboard when the Argus/200 smart card acceptance device is used since this reader does not have a keypad. This makes this reader only slightly less secure since the PIN travels in the clear along the keyboard cable into the workstation or client machine.

2. He can use the TSS Signature Verification feature. He signs his name with a special pen which measures the dynamics of his signature as he signs. The data collected from the pen is compared with reference data stored on the user′s smart card. The reference data is encrypted under a session key when transmitted from the card.

   Signature verification is more secure than the use of a PIN, but it requires the extra expense of the signature verification hardware.

   The card design will permit any external biometric verification method to be used as long as it follows the interface conventions used with signature verification today.

The card counts how many consecutive times each user fails verification, either by PIN or by signature. Each card user is assigned a limit for how many times he can fail, and he is locked out of the card if he exceeds that limit. The limit can be anywhere from 0 through 255, where a value of zero means that the user has no limit at all on verification failures.

Note: In the current design, the Signature Verification is not implemented in the Secured Single Signon solution but can be added to enhance the security of this solution.

4.2.5 User Verification Commands

The following commands are used in user verification.

- The *Verify PIN* command is used to verify a user′s PIN.
- The *Set User* command is used to inform the card that a user has verified successfully using signature verification. The signature verification processing is done outside the card, so the card must be notified of the results. The command is executed using a secure, encrypted transaction.

- The *Increment Verification Failure Count* command is used to increment the count of consecutive verification failures when the user fails signature verification.
- The *Reset Verification Failure Count* command is used to clear the count of verification failures when the user has been locked out. This command is typically restricted so that only high-security users can issue it.

4.2.6 General Commands

Several commands do not fall into any particular category. These include:

- The *Read Status* command is used to read miscellaneous status information from the card.
- The *Read Device Information* command is used to read the card serial number, part number, and other similar information.
- The *Reinitialize Device* command restores the smart card to the state it had when originally delivered to the customer. All data the customer has loaded to the card is destroyed. Since the smart card is a security device, it is possible to configure it so that no user can execute any useful command; in this case, Reinitialize Device is a convenient way to recover.

4.2.7 Data Block Commands

Data Blocks are customer-defined data areas in the EEPROM of the Personal Security card. They are described in detail in 4.8, “Data Blocks” on page 29. The following commands are used to manage and process data blocks.

- The *Create Block* command allocates a new data block on the smart card.
- The *Delete Block* command removes a data block from the card.
- *Read Block* reads data from a data block.
- *Write Block* stores data in a data block.
- The *Query Block* command provides information about a data block.
- The *Get Block Directory* command returns a list of data blocks that are defined on the smart card. It also returns the amount of free space remaining for new data blocks.
- The *Clear Block Area* command deletes all data blocks from the card in a single operation.

4.2.8 IML Commands

As mentioned on 4.1.1, “Personal Security Card Security Features” on page 20, new code can be added to the smart card using the IML features. This code might be patches to existing functions, or entirely new functions that were not part of the original design. IML is typically performed as part of the smart card manufacturing and test process.

A simple set of commands are used for IML.

- The *IML Enable* command activates *IML Mode* in the smart card. This is a prerequisite to any of the other IML commands. As part of the IML Enable process, the card destroys *all* secret data in the EEPROM. This guarantees that a new program loaded into the card will not have access to any secret data left by the previous application, including keys and data blocks.

- The *Load Serial Number* command is used to load the smart card serial number. This is only allowed one time; subsequent attempts to load a serial number will be rejected.
- The *Test EEPROM* command can be used to selectively test portions of the EEPROM memory.
- The *Load Data* command is used to download the new microcode or data into the EEPROM memory.
- The *End IML* command exits IML mode and recalculates all pointers to account for the data which was loaded.

4.3 Secure Sessions

A secure session can be established between the Personal Security Card and any other cryptographic device. In TSS, sessions are established between the smart card and the 4754 reader, and between the smart card and the 4755 cryptographic adapter.

Secure session establishment is a process which results in a common, randomly derived session key shared between two devices. In order to establish the session, either the two devices must share a common Key Encrypting Key, or a third device must share Key Encrypting Keys with each of the two devices establishing the session.

The session key will be different each time a session is established. For example, the smart card and reader establish a new session key each time the card is inserted. Because of its limited lifetime, the session key provides a considerable amount of security.

The session key is primarily used in two ways:

1. To protect information transmitted across the interface between two devices. Tapping the line will not expose the information if it is encrypted under the session key.
2. To limit functions to devices that have sessions established with each other. If the request or response data for a command is encrypted under a session key, the request or response can only be interpreted by a device that shares the session key.

Notes:

1. Since the Argus/200 smart card acceptance device does not contain a cryptographic processor, you need a 4755 cryptographic adapter to establish a secure session between the Personal Security card and the workstation or client machine.
2. If employees need to access systems from a workstation or client machine other than their normal one, it is better not to include the secure session in the Secured Single Signon set up. This prevents additional key management since you don′t need to keep the needed keys for the secure session in the cryptographic adapter in the participating workstations or client machines.
3. For additional security reasons, the card initialization program requires a secure session between the environment where the program is running in and the Personal Security card. That implies that a least one 4754 reader

together with one 4755 cryptographic adapter is available for card initialization in the administrator's workstation.

4.4 Authorization - Overview

Authorization in the Personal Security card consists of two things:

- Authorization to execute commands
- Authorization to access data areas

Three things work in concert to define a user's authorization:

- A *User Profile* defines the current card user.
- *Command Configuration Data* defines the state and prerequisites needed to execute each command.
- *Data Block Headers* define the requirements that must be met to access each data block on the card.

Each of these is described in the sections that follow.

4.5 User Profiles

Each smart card user has an independent user profile to define the user's authorization and other personal information. Four profiles are stored on the Personal Security card; the four users are designated **User 0** through **User 3**. This allows up to four completely independent users, each with their own authorization data, to share a single Personal Security card.

The user profiles are summarized in Figure 6. Each of the four profiles contain the following fields:

- An 8-byte *User ID*, much like the user ID on a computer system. It is used to identify the user.

  Note: In a Secured Single Signon solution there is no relation between this user ID and the user ID that is used in the signon to the server application. The user ID that is used in the signon is stored in cryptographically protected storage on the SSS Card together with the server application name and a pointer to the secret application key.
- An 8-byte *PIN* (Personal Identification Number) which is used to verify the user's identity. As mentioned earlier, signature verification is available as an alternative to the use of a PIN.
- The user's *Authority Level*. This is a value between 0 and 255 which defines the authority level of one user relative to the others. A higher number means greater authority. The Authority Level is used in conjunction with fields in Command Configuration Data and Data Block Headers, which will be described later.
- A set of 256 *Command Authorization Flags*. The Personal Security Card can have a maximum of 64 commands in its repertoire. Other TSS components can have up to 256 commands. The Command Authorization Flags contain one bit for each of the 256 possible commands. If a bit is ON, the corresponding command is permitted for that user, although other

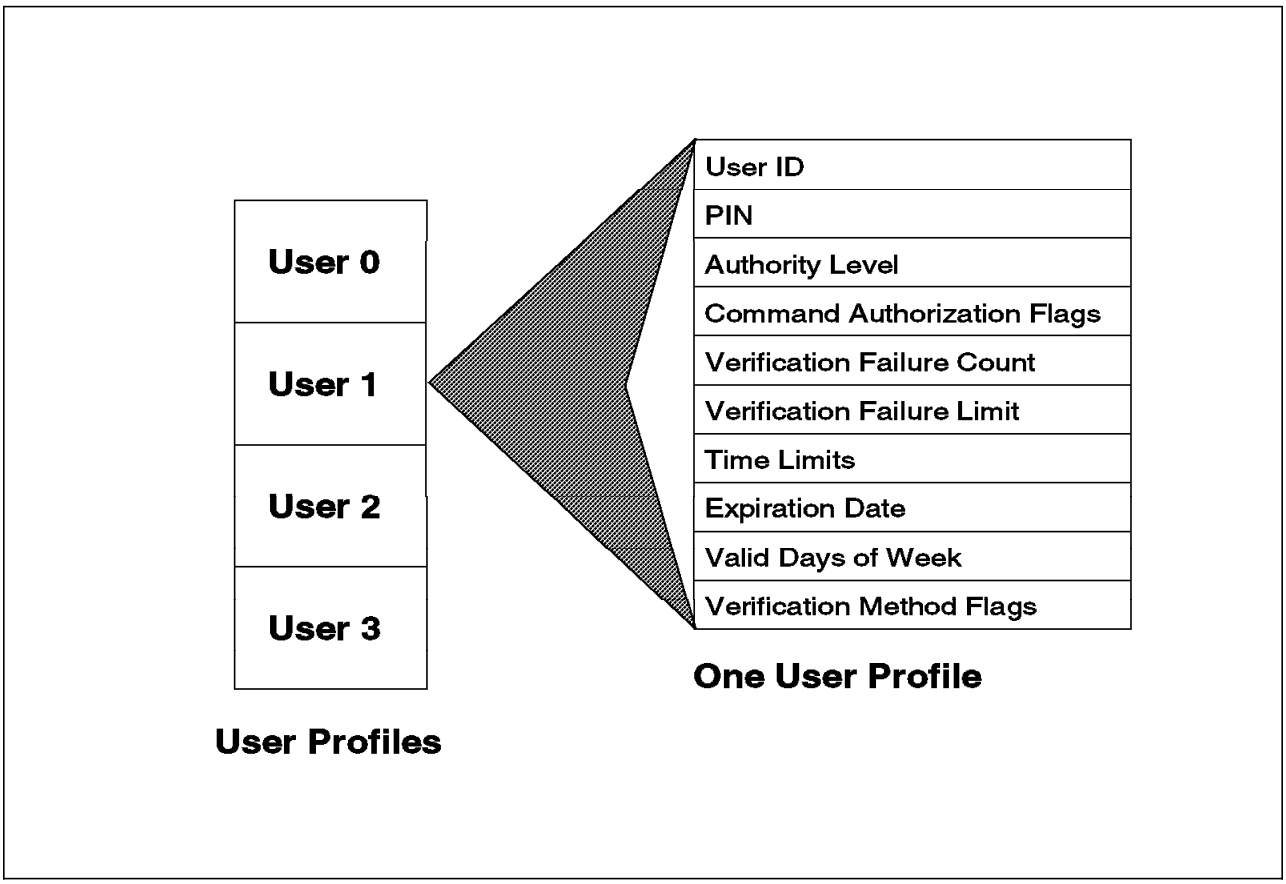

Figure 6. User Profile Overview

authorization controls may prohibit it. If the bit is OFF, that command is *not* permitted for that user.

- Consecutive verification failures by PIN or signature verification are counted in the *Verification Failure Count* field. The maximum number of consecutive failures allowed for each user is assigned in his *Verification Failure Limit* field. Each of these can range from 0 through 255. For the Verification Failure Limit field, a value of zero has a special meaning: that user is allowed unlimited failures.

  Once a user reaches his failure limit, he will not be able to use the card until another user with sufficient authority resets his failure count.

- Several fields define *when* the user is permitted to use the smart card. The *Time Limits* define the earliest and latest times of the day for card use. The *Expiration Date* defines the last date the user will be allowed to use the card. The *Valid Days of Week* define which days (Sunday through Saturday) are valid for the user. As an example, a user might be allowed to use the smart card only between 8:00 AM and 5:00 PM on Monday through Friday.

  Time limit, day of week, or expiration date checking can be disabled for individual users if desired.

- Two flags, the *Verification Methods Flags,* define how the user's identity is to be verified. There are three possibilities.

  – Verify the user only with PIN verification.

  – Verify the user only with signature verification.

– Verify the user with signature verification if it is available, otherwise verify using PIN.

Note: In the current design of the Secured Single Signon solution, some of the fields in the user profile are initialized to the default values, which do not provide the level of security that would be possible if they were set to other values.

4.6 Command Configuration Data

There are two bytes of Command Configuration Data associated with each command the smart card can execute. This data describes the security characteristics of each command. The interaction between a User Profile and the Command Configuration Data determines whether a given user is authorized for a given command.

The Command Configuration Data contains the following fields.

- A *Command Unavailable* flag, if set, logically removes the associated command from the smart card command set.
- If the *Secure Session Required* flag is set for a command, that command will only be accepted if the sender has a secure session in effect with the smart card. A secure session is impossible if the other device does not share a secret key with the smart card.
- Two flags are available to restrict commands according to the verification status of the user. If the *Initial Verification Required* flag is set, the command will be rejected unless the user has verified his identity with a PIN or a signature. If the *Pre-Execution Verification Required* flag is set, the user is required to reverify his identity *every time* he executes the command. This guarantees the user is valid and is actually at the terminal when the command is used.
- The *Time Limit Enable* flag indicates whether time, date, and day of week checking should be performed for a command. For low security commands, it may not be important to restrict when they can be executed.
- Two fields restrict commands according to the Authority Level in the user′s profile. The *Required Authority Level* defines the authority level (0-255) the user must have to be authorized for the command. The *Authority Exact Match* flag defines how the authority level is checked. If Authority Exact Match is *not* set, the command will be accepted if the user′s authority level is *greater than or equal to* the Required Authority Level. If Authority Exact Match is set, however, the command will only be accepted if the user′s authority level is *exactly equal* to the Required Authority Level.

4.7 Other Authorization Features

The following are some additional authorization features built into the Personal Security card.

4.7.1 Universally Authorized Commands

A small subset of the smart card commands are *always* authorized, regardless of the profiles and Command Configuration Data. These are commands every user will need in order to operate the system, and they are not related to security in any way. These are called the Universally Authorized Commands.

4.7.2 Holiday Table

To further restrict when the smart card can be used, it contains a Holiday Table which can be programmed with dates the customer expects his office to be closed. On the dates programmed into this table, none of the four card users will be permitted to use the smart card. Typically, this table would be loaded with dates such as Christmas, New Years Day, and various national holidays. The restriction prevents an adversary from attempting to break the system on a day when nobody else would be in the office to see him.

4.7.3 Transfer of User Profile

The smart card has a very powerful authorization feature which is used by the other TSS devices. The 4755 Cryptographic Adapter and the 4754 Security Interface Unit use the same authorization approach as the smart card, but they have an additional user profile known as the *Guest Profile*. When the smart card is inserted in the reader and the card user verifies his identity, his user profile is downloaded to the Guest User Profile in the 4754 and 4755. Thus, the authorization information carried in the user's smart card is used in a very powerful way. When inserted at the workstation, the portable authorization data from his smart card is downloaded and *controls authorization in the workstation itself*. This is a very powerful concept. It means that two users can have completely different capabilities at the same workstation, based on the data in their smart cards.

For security reasons, the profile is only downloaded to another device if that device has a secure session in effect with the smart card. The data is encrypted under the session key when transferred, so that it cannot be altered, and its contents are not exposed.

4.8 Data Blocks

Data Blocks provide a general purpose data storage system on the smart card, much like files on a diskette. The blocks can be created, deleted, read, and written. There are utility functions to get information about a specific block, to get a directory of all blocks on the card, and to delete all defined blocks in a single operation.

Each data block consists of two parts, a 21 byte *header*, immediately followed by the block's *data area*. The data area can be from 8 through 2040 bytes, and is fixed in size once the block is allocated.

Figure 7 summarizes the layout of data blocks in the EEPROM memory.

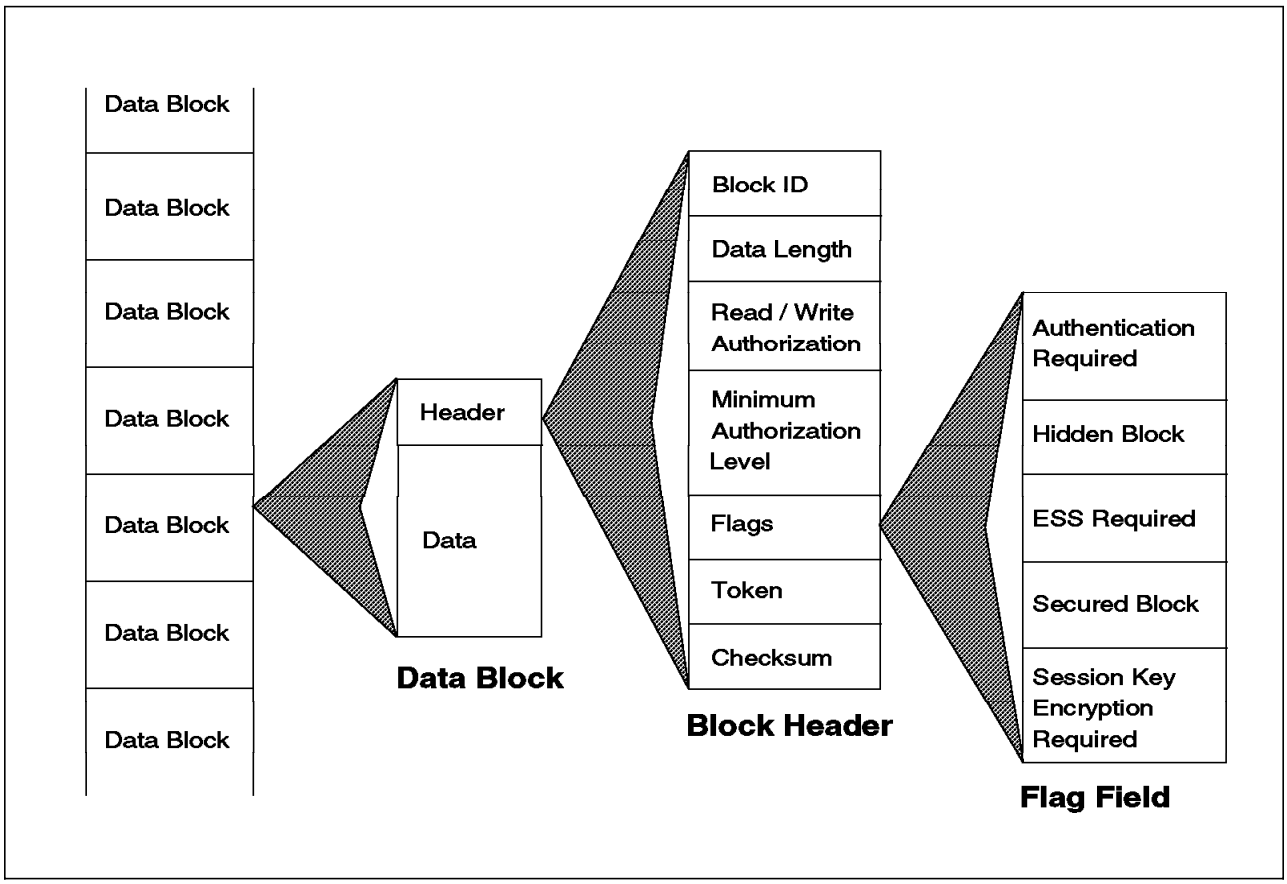

Figure 7. Summary of Data Block Layout in EEPROM

4.8.1 Block Header Contents

The block header contains information for identifying the block, as well as control and security information. It contains the following fields:

- The *Block ID* is an 8-byte ″name″ for the data block, similar to a file name.
- The *length* of the block′s data area is stored in the header.
- Each block has a *Block Token*, which is a ″password″ for the block. The token must be given in order to access the block. If the user does not know the token, he cannot access the data block.

  By convention, a token of all zeroes is used when the application does not need this level of security.

  Note: In the Secured Single Signon solution, the token for the data blocks that contains the security sensitive information is a MAC that is generated outside the SSS Card during card initialization. Input for this MAC is the information that will be stored in this data block. The secret application key is used as MAC key. The secret application key that is stored on the SSS Card can only be used to perform a MACVER operation. This prevents the user from reading or updating the data blocks.

  Whenever a PassTicket is generated, the smart card will first verify (internally) the MAC which is used as the block token. If the MAC calculated on the internal data is not the same as the block token, the PassTicket generate command will abort with an error.

- A set of *Read/Write Authorization Flags* is associated with each block. This field contains one read authorization and one write authorization flag for each of the four card users. The user can read the block only if his read authorization bit is set in the header, and can write to the block (or delete it) only if his write authorization bit is set. Each user can have any combination of access to a data block: none, read-only, write-only, or read-write.

  Note: Only the administrator profile has write access to the PassTicket data blocks that are initialized on the SSS Card.

- The *Minimum Authority Level* defines the authority level the user must have in his profile in order to be granted access to the data block. If his authority level is less than this value, he cannot access the block in any way.

- An *Authentication Required* flag can be set to exclude access by any user who has not verified his identity. Successful PIN or signature verification must precede any attempt to access such a block.

  Note: This flag is set in the Secured Single Signon solution. That means that only authenticated users can access the data blocks that are needed to generate a PassTicket.

- Setting the *Hidden Block* flag will keep the block from being listed when the Get Block Directory command is executed. This is similar to hidden files on a PC, which are not listed with the **DIR** command.

- If the *Secure Session Required* flag is set in the block header, no attempt to access the block will be accepted unless the sending device has a secure session in effect with the smart card. Only devices that share a common key with the smart card can establish a secure session. Thus, blocks with this flag set can only be accessed by an authentic device the customer has programmed with the correct keys.

- A feature is available that will prevent access to the block unless the user knows a secret data key stored on the smart card. When the *Secured Block* flag is set, the Block Token must be encrypted under the specified data key when it is sent to the card. The card decrypts the token, then compares it with the token stored in the block header. The tokens will not match unless the correct key was used.

  This has the same effect as encrypting the data in the block itself; it prevents anyone from reading or writing useful data unless they know the encryption key. In addition, it prevents anyone from overwriting your block with meaningless data. Instead of incorrect data being written because of the wrong key, this approach keeps data from being written at all. It also results in much less encryption overhead than encrypting the data in the block.

- The *Session Key Encryption Required* flag can be set to keep the data secret when it is read from or written to the block. When this flag is set, all data read from the block will be encrypted under the secure session key established with the requesting device. The card will expect all data written to the block to be encrypted with the same key when it is received.

  Encryption under the session key is available as an option of the read and write commands, but this flag makes it a *requirement* for the block; reading or writing data in the clear will not be allowed.

- Finally, each data block header contains a *Checksum*. The checksum is used to verify that no memory problem has corrupted the data in the block. It is calculated each time the block is written, and verified each time the block is read.

4.8.2 How the Data Blocks are Used

Data blocks are very flexible and lend themselves to a wide variety of uses. On the SSS Card, relevant input information for the PassTicket generate algorithm is stored in various data blocks on the card. One data block contains the list the applications the user is allowed to access.

From a data block, there is a pointer for each application into a table that is composed of a series of data blocks. Each data block in this table contains the user ID that needs to be used for this application, the application name and a pointer to the key register that holds the secret application key.

Each data block in the table is cryptographically protected to prevent the user from changing the relation between the application name, the user ID that will be used and the secret application key for that application.

Appendix C, “Applications of the Personal Security Card” on page 103 describes some other ways data blocks are being used in Personal Security card applications.

4.9 SSS Card

The SSS Card is a special version of the IBM Personal Security card. The PassTicket algorithm is implemented on the card. Some, less frequently used, functions are removed to free up EEPROM memory to allow the installation of the PassTicket algorithm. Of particular interest is the lack of the Key_Export and KEK_Export commands. This is an extra assurance that the secret application keys inside the key registers cannot be exported from the SSS Card and used in an external cryptographic facility.

Special card initialization programs are needed to initialize the data blocks on the cards and to load the user specific information into the various data blocks.

The following specialized data blocks are initialized on the SSS Card:

- One data block is used to store the authorized applications together with the internal pointer: blkname.
- A series of data blocks is used as a table of application blocks.
- Key registers are used to store the secret application key.

The SSS Card needs to be personalized. The data blocks on the SSS Card are loaded with security sensitive information. The applications the user is authorized to use and the user ID that needs to be used for that application is stored in these data blocks.

The solution packages provides sample programs that can be used to design card initialization programs for different administration workstations on different platforms, such as OS/2 workstations or AIX RISC/6000 client machines.

The initialization programs also provide the normal TSS component initialization to load the following data on the card:

- User profile
- Command configuration data

Note: It is vital that the command configuration entry for the PassTicket generation command has the *Initial Verification Required* bit set to **ON**.

- Master Key and Session Key

The standard *Hardware Initialization and Key Management Utility* (HIKM) is used to perform the TSS component initialization. The solution package contains a sample input file for the HIKM utility.

4.9.1 Directory Block

Figure 8 depicts an overview of the directory block.

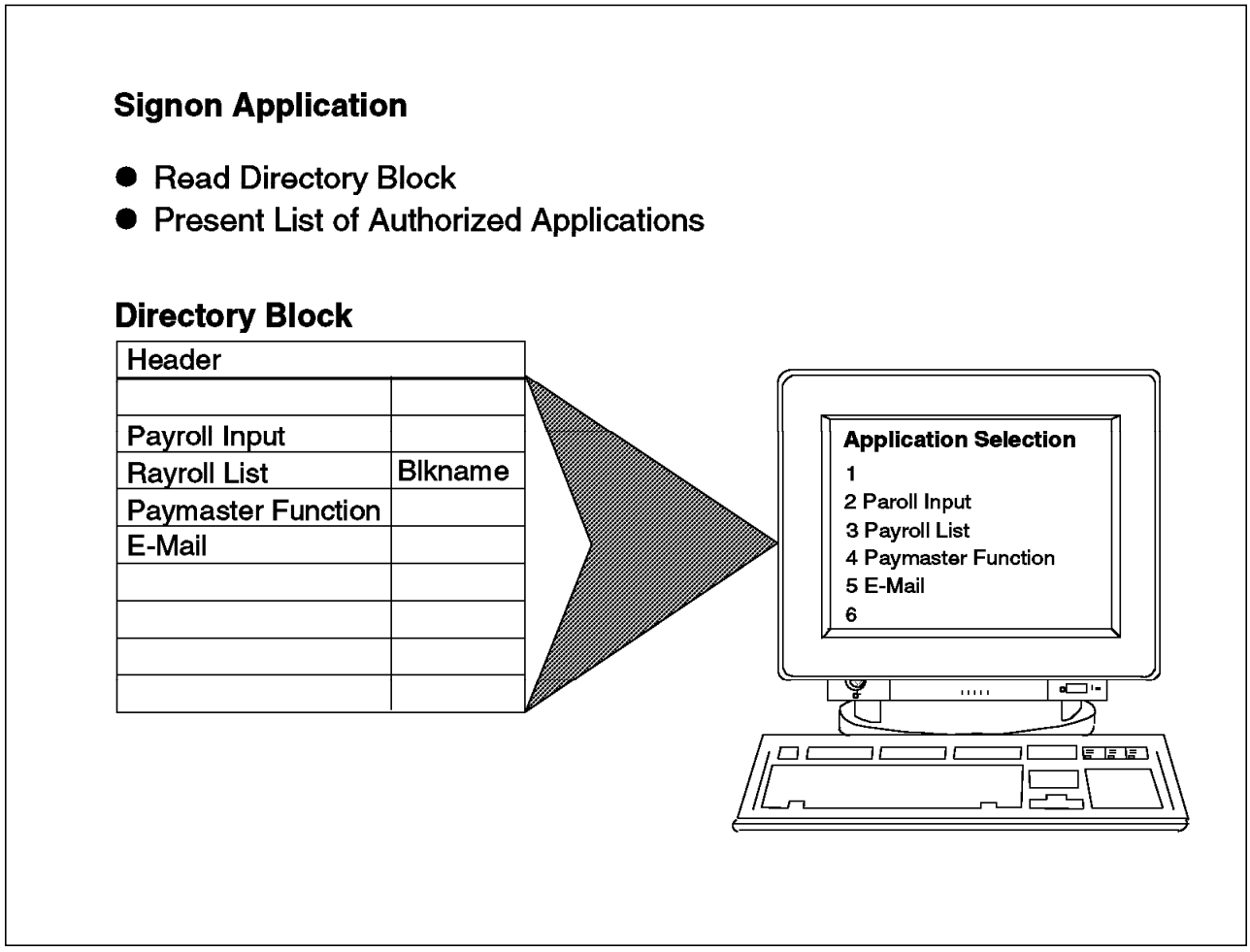

Figure 8. Overview of the Directory Block

The Directory Block (DIRBLOCK) contains some header information and a list of authorized applications and a pointer to a *Application Data Block*. The fields in this block are of variable length, and separated from each other by the standard C-Language end-of-string character "\0."

There is no security sensitive information is this data block.

Once a user is identified and authenticated by entering his correct Personal Identification Number, the signon application reads the directory block from the card and displays the list on the screen of the workstation or client machine. The pointer to the application data block is retained by the signon application and used in the call for the PassTicket generate routine.

4.9.2 Application Data Block

The Application Data Block (APLBLKxx) contains:

- The application ID (8 bytes)
- The User ID for that application (8 bytes)
- Pointer to a key register that holds the secret application key. (8 bytes - divided into a 1 byte register, right-aligned, and padded on the left with space characters)

For each entry in the DIRBLOCK, an application data block, APLBLKxx, is initialized, where “xx” is an integer number with leading zeroes. The xx is also the register number. So, viewing the block names tells you which registers are used.

4.9.3 Key Registers and Control Vectors

Key values of the secret application key are stored in the key registers together with the *control vectors* for the specified keys.

The control vector that is associated with the secret application keys prohibits the export of the keys and the key usage in normal cryptographic commands, specifically a MAC Generate.

Note: Cryptographic commands use a set of distinct key types. The combination of the command set and the TSS capability to separate keys into different types provides a secure cryptographic system that blocks many attacks that can be directed against a cryptographic system.

The TSS products, including the Personal Security card, use a control vector to separate keys into distinct key types and to further restrict the use of a key. A control vector is a nonsecret value that is stored together with the key and is cryptographically associated with the key by being exclusive ORed with a *master key* or a *Key Encrypting Key*.

If a control vector that accompanies a key is different from the control vector that is used in the cryptographic operation, the correct key cannot be recovered.

4.9.4 SSS Card Data Block Protection

During SSS Card initialization, a Message Authentication Code is generated before the information is stored on the SSS Card. Input for this MAC are the three data fields:

- Appl_ID
- User_ID
- Key register pointer

The secret application key is used as the MAC generate key. This MAC is stored in the *token field* of the *block header* of the application data block.

The SSS Card will internally verify the MAC in the token field of the block header before the algorithm on the SSS Card can generate a PassTicket. The SSS Card uses the secret application key that is stored in the data key tables on the smart card to verify the MAC.

This technique prevents the misuse of the SSS Card. The internal verify will fail if an attacker changed any of the fields within an application data block. The

attacker cannot generate a new MAC after he changed one of the fields since he has no access the secret application key.

Figure 9 depicts an overview of the data blocks on the SSS Card.

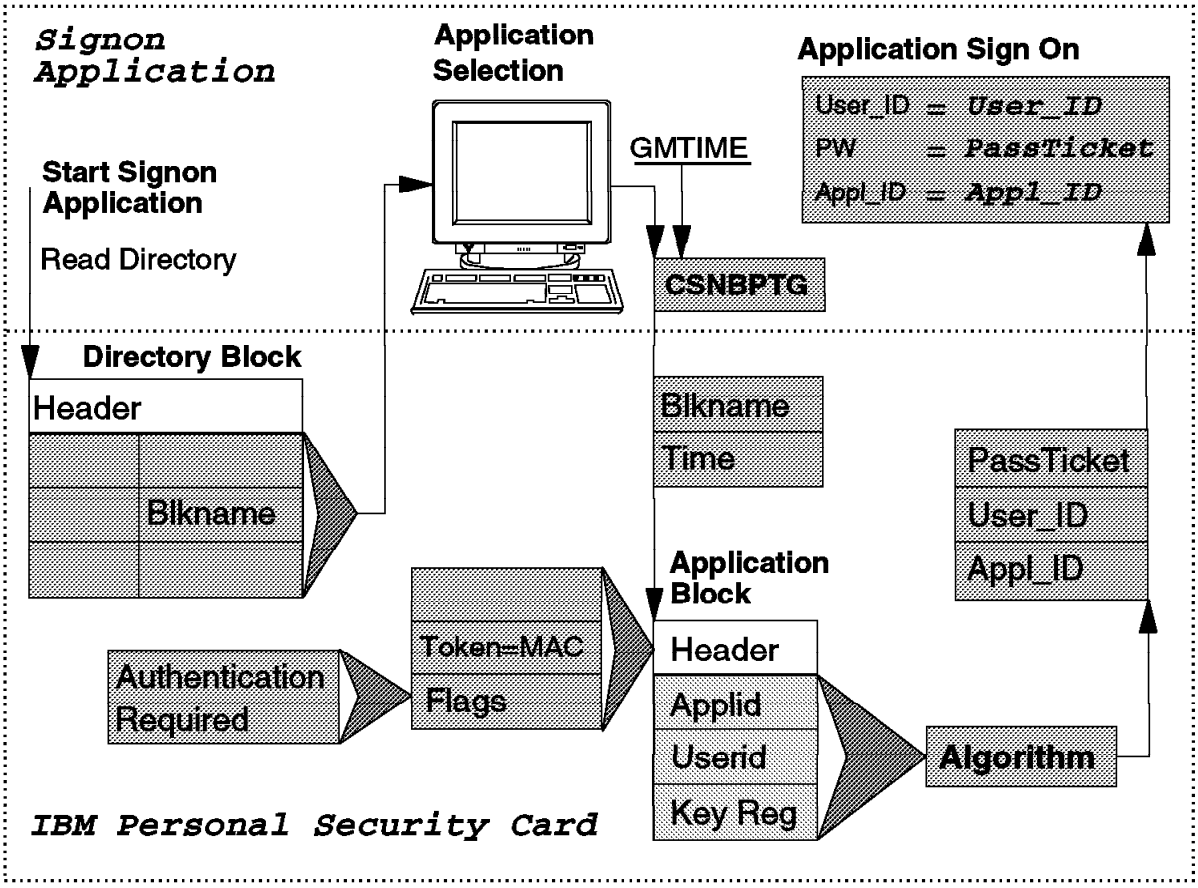

Figure 9. Overview of the Data Blocks on the SSS Card

4.10 SSS Card Initialization

There are several layers of initialization of the SSS Card.

1. Microcode load, which is part of the manufacturing process of the SSS Card. This process will not be discussed in this document.
2. Normal TSS component initialization, where things like:
   - Configuration Data
   - User Profiles
   - Master and Session Keys (KEKs)

   are loaded onto the SSS Card.
3. Application Initialization, which writes data blocks and key registers on the SSS Card.

4.10.1 Component Initialization

The component initialization is performed by using the standard *Hardware Initialization and Key Management* (HIKM) utility. HIKM is a component of the TSS software and should be performed by the security administrator of the installation. A sample HIKM input file is part of the solution package.

A user profile for the administrator will be initialized. His PIN is loaded on the card. Also the session key that is needed to establish a secure session between the security interface unit and the SSS Card is loaded on the card.

After the component initialization, the cards are loaded as identical cards. That does not cause any security exposures since there is no customized security sensitive user data on the cards yet. However, it is vital that the new command configuration entry for the PassTicket generate routine (X'33') has the *Initial Verification Required* bit set.

4.10.2 Application Initialization

The application initialization writes the data blocks and key registers on the SSS Card.

A sample program to load these data blocks is part of the solution package. This sample program assumes a host environment with RACF protected server applications. Access information is extracted from the RACF data base and sent to the workstation of the security administrator that is initializing the SSS Cards.

The sequence of events in this card initialization program is as follows:

1. The initialization program requests the administrator to insert the SSS Card in the 4754 Security Interface Unit. The program executes a *Card_Contact* (CSUASCK) verb to initiate communication with the Personal Security card. Among other things, this command will establish a secure session between the security interface unit and the SSS Card.
2. A *Cryptographic_Facility_Query* (CSUACFQ) verb is executed to obtain the hardware status and profile information from the SSS Card This command determines the presence of a Personal Security card in the security interface unit.
3. A *Profile_Activate* (CSUAAPR) verb is executed to activate the user profile for the security administrator on the Personal Security card. The activate profile verb also allows the Personal Security card to accept a PIN for that profile.
4. The security administrator is now prompted for his PIN. The PIN transfers directly from the security interface unit to the SSS Card without passing through the workstation.

The following steps assume that an output file with the extract information from the RACF data base is available. This file contains information about the applications the user is authorized to use and the user ID that should be used for these applications. It also contains the secret application key for these applications.

5. A *Card_Block_Allocate* (CSUASCA) verb is used to create the *Directory_Data_Block* on the SSS Card. The block name is DIRBLOCK, and the token for this block is also DIRBLOCK.

   This verb also makes an entry for the directory data block in the directory of the SSS Card.

6. A *Card_Block_Write* (CSUASCW) verb is used to write information from the file with the extracted RACF information into the directory block. Pointers to the *Application_Data_Block* are created according to the naming standards.

In order to build the next data block that need to be stored on the SSS Card, some preparation needs to be performed in the workstation.

For example, the control vector for the secret application needs to be built and the MAC for the application data block needs to be calculated.

7. The *Control_Vector_Generate* (CSNBCVG) verb is used to create the control vector that will be associated with the secret application key. At this point in time, the secret application key needs to be enabled to be used in a MAC operation.

8. A *Key_Token_Build* (CSNBKTB) verb is used to assemble a key token. This key token will be used to store the secret application key together with the control vector that is built in the previous step.

   Note: A key token is a 64-byte data structure that includes the key and other information, such as the control vector, that is frequently needed when the key is needed.

   There are various key tokens that serve different environments. For example, an *external* key token contains an external key that is associated with a control vector and a key-encryption-key. An *internal* key token contains an operational key that is associated with a control vector and the master key of the cryptographic device that is used.

9. Several key management verbs are performed now to prepare the usage of the secret application key in the MAC operation.

   - A *Key_Part_Import* (CSNBKPI) verb is used to store the clear secret application key in the specified key token and associate the key with the specified control vector. The key token and the control vector are created in the previous steps.
   - A *Key_Export* (CSNBKEX) verb is used to build an external key token. It copies information from the specified internal key token that was created in the previous step.
   - A *Key_Import* (CSNBKIM) verb is then used to create an internal key token to be used in the following step. The secret application key is in operational form (associated the master key of the workstation and the control vector) available in the new internal key token.

10. A *MAC-Generate* (CSNBMGN) verb is used to generate a MAC for the following user data that is extracted from the RACF data base:

    - Application ID
    - User ID that should be used for that application

    The key register that is used for this data block is added to this user data information.

11. A *Card_Block_Allocate* verb and the *Card-Block_Write* are again used to allocate and write the application data block on the SSS Card. The header information for the block is as follows; the block name is determined by the naming standards and the token is replaced by the MAC that is calculated in the previous step.

    The permission bits are set in a way that the user profile for the security administrator is authorized to write (update) this data block. Other user profiles has only read access. Also, the *verification_required* bit is set.

12. The *Control_Vector_Generate* (CSNBCVG) verb is used again to create a control vector that will be associated with the secret application key. At this point in time, the control vector should prevent the usage of the secret application key on the SSS Card in a MAC or Key_Export operation.

13. Key management verbs are used to prepare the secret application key to be exported to the SSS Card. A *Key_Import* verb is used to store the secret application key with needed control information on the SSS Card.

14. As a final test, a PassTicket is generated to see if the data blocks are accessible on the SSS Card. The PassTicket is not used to sign on at this time. The end user will be requested to test the card in his own environment.

15. At the end of the initialization, a *Card_Eject* (CSUASCE) is used to eject the SSS Card from the security interface unit.

Chapter 5. Workstation Environment

This chapter provides a high-level description of the workstation hardware and software products that were used to design and test the Secured Single Signon described in this document.

5.1 Hardware

In the context of this project, *workstations* are personal computers running versions of DOS. Each system must contain an adapter suitable for the 3270 emulator program and connection used. This adapter might be a “3270 emulator” adapter (with a 3270 coax connector), a LAN adapter (using a “3270 gateway” function installed in another system on the LAN), a “SNA” or “multifunction communications adapter” connected to a modem, or any other adapter and interface supported by DOS and the 3270 emulator program used.

A serial port (with a 25-pin D-shell connector) is also required for connecting the smart card reader to each personal computer.

The security administrator who prepares smart cards for users requires additional hardware features in his personal computer. He requires the IBM 4755 Cryptographic Adapter, and the IBM 4754 Security Interface Unit. These two elements are required only in the administrator′s system.

As a general term, “workstation” sometimes implies a more powerful system than the average personal computer. For example, it sometimes implies a high-performance UNIX** system that is used as a client machine. Although client machines are not tested as part of this project, it is still possible to implement the design principles as described in this document in UNIX or AIX based client machines.

5.2 Workstation Operating System

The specific test described in this document is for DOS: IBM′s PC DOS or Microsoft′s** MS DOS. Our testing was done with DOS Releases 5 and 6. The specific programming and operational details relate only to a simple DOS environment, with a 3270 emulator which provides the HLLAPI interface (described below).

With minor changes, our solution works in Microsoft′s Windows** 3.1 and IBM′s OS/2 environments -- provided suitable 3270 emulators are available. (Some of the development work used the “DOS box” environment of OS/2). The minor changes relate to hot-key assignments, emulator session names, drivers for the Argus/200 smart card acceptance device, and so forth.

5.3 3270 Emulators

The Secured Single Signon test described in this document assumes that a *3270 emulator* program, running in a personal computer, is used for host communication. As the name implies, this program “emulates” an IBM 3270 terminal; that is, the host system is structured to work with a “real” IBM 3270 terminal and is unaware that it is not working with a “real” IBM 3270 terminal.

Many 3270 emulator programs are available, from IBM and other vendors. Our environment *requires* a 3270 emulator that supports HLLAPI (or EHLLAPI or EEHLLAPI). HLLAPI, a programming interface to 3270 emulator programs, is briefly described in the next section. Our test is independent of the connection method between the personal computer and host. The personal computer could use a basic 3270 (coax) adapter, a LAN connection, an SNA connection, or any other connection supported by the emulator and host.

We used the IBM 3270 emulator available with OS/2 2.1 (running in a DOS "box") and its associated EHLLAPI interface for program development and testing. At other times we have also used the IBM Personal Communications/3270 emulator and several older emulators, all under DOS.

Our design, as described in this document, is built around DOS (PC-DOS or MS-DOS) and DOS-based 3270 emulators. These emulators are forms of Terminate and Stay Resident (TSR) programs. When the emulator program is started, it subtracts the memory it uses from what is available to DOS, and returns control to DOS (in the smaller memory area). The user can switch between the DOS session and the emulator session by using a "hot-key" combination. (The most common hot-key combination is Alt-Esc.)

Normal DOS programs (with a few restrictions concerning communications programs) can be run in the DOS session while the emulator is maintaining a 3270 session with a host. When the user "hot keys" from the DOS session to the emulator session, any DOS program is suspended. When the user "hot keys" from the emulator session back to DOS, the DOS program (if any) resumes, but the emulator session is *not* fully suspended. The emulator responds to 3270 control signals from the host and generally maintains the existence of the 3270 session with the host.

Personal computer keyboards do not match IBM 3270 keyboards. One function of an emulator program is to "map" the personal computer keyboard to provide 3270-like functions. This is usually done with a control file, and can be changed by the user. (One challenge in using an emulator is to remember or discover the key mapping for your session. Which personal computer keys represent the 3270 PA1 or CLEAR keys?) The emulator program also provides the necessary EBCDIC/ASCII translations necessary to perform the 3270 emulation.

Most IBM 3270 emulator programs support multiple 3270 *sessions*. The test we describe can use multiple sessions, although some planning is necessary to associate specific host applications with specific emulator sessions.

Practically all 3270 emulator programs provide utilities to send files from DOS to the host and to receive files from the host and store them in DOS. Many emulators provide SEND and RECEIVE commands (which operate in the DOS session) for this purpose; these often work in conjunction with the IBM-provided IND$FILE program in the host MVS or VM system.

While our project (and this document) are in a DOS context, all of the functions and many of the comments can be applied to OS/2 and MS Windows contexts, with only minor changes.

5.4 HLLAPI

HLLAPI is an acronym for High-Level Language Application Program Interface. HLLAPI provides a programming interface for 3270 (and other) emulator programs. Several HLLAPI implementations are available (as parts of several different 3270 emulator products). Some versions are called EHLLAPI or EEHLLAPI. There are minor differences among the various HLLAPI, EHLLAPI, and EEHLLAPI implementations; these differences are not relevant for the work described in this document and will be ignored.

In general, HLLAPI is provided by a TSR module. This module communicates with the 3270 emulator session and the DOS session. It provides a way to use the emulator session under *program* control instead of human-and-keyboard control. Using the HLLAPI interface, a DOS program can simulate keyboard for the 3270 emulator session and "read" the emulated 3270 screen (which is maintained, in memory, by the emulator program).

For example, a HLLAPI program (running under DOS) could search the emulated 3270 screen (in memory) for the string ″ENTER LOGON:″, simulate keying a user ID and password, and simulate pressing the ENTER key. The emulator program would send the user ID and password to the host, and the host would (presumably) process the logon and start an application. The HLLAPI program could wait for the host response, examine it, and simulate more keystrokes for entry. The HLLAPI program could also terminate. In this case, the 3270 session is not affected. The user could "hot key" to the 3270 session and continue the session.

The HLLAPI interface is quite high-level, and can be used from several languages (C, BASIC, COBOL, PASCAL). A small language interface module (LIM) connects programs written in a particular language to the HLLAPI module. The appropriate LIM is linked with the application program.

There are about 45 HLLAPI functions, although typical programs use only a few of these. All HLLAPI function calls have the same format (shown here in C):

```
hllc (&function, datastring, &datalength, &returncode);
```

where *function* is a number specifying the requested action, *datastring* is a data area (input or output string), and so forth. Commonly used functions include:

- Connect Presentation Space
- Copy Field to String
- Copy String to Field
- Pause
- Search Field
- Send Key

and similar operations. Action keys, sent with the Send Key function, are preceded with the "@" symbol. For example, "@E" means simulate pressing the ENTER key, "@C" means simulate pressing the CLEAR key, and so forth.

HLLAPI programming tends to be simple for an experienced programmer. In practice, HLLAPI applications usually depend on the exact format and content of 3270 data screens sent from the host. Minor changes in screen content or format can cause HLLAPI applications to fail, or at best, to require continuous maintenance.

HLLAPI implementations are available with 3270 emulators for OS/2, MS Windows, and other operating systems. Operational details differ; for example, there is no equivalent to a TSR under OS/2. Programming compatibility across platforms is good, although minor adjustments should be expected when changing platforms or emulator products.

5.5 TSS Server Environment

The Transaction Security System Server Environment consists of software and Transaction Security System I/O components. An important characteristic of the Transaction Security System hardware is a *tamper-resistant environment* in which access-control decisions can be enforced, cryptographic processing can be performed, and Personal Security card data can be stored.

Figure 10 shows the server environment that exists in the TSS smart card implementation.

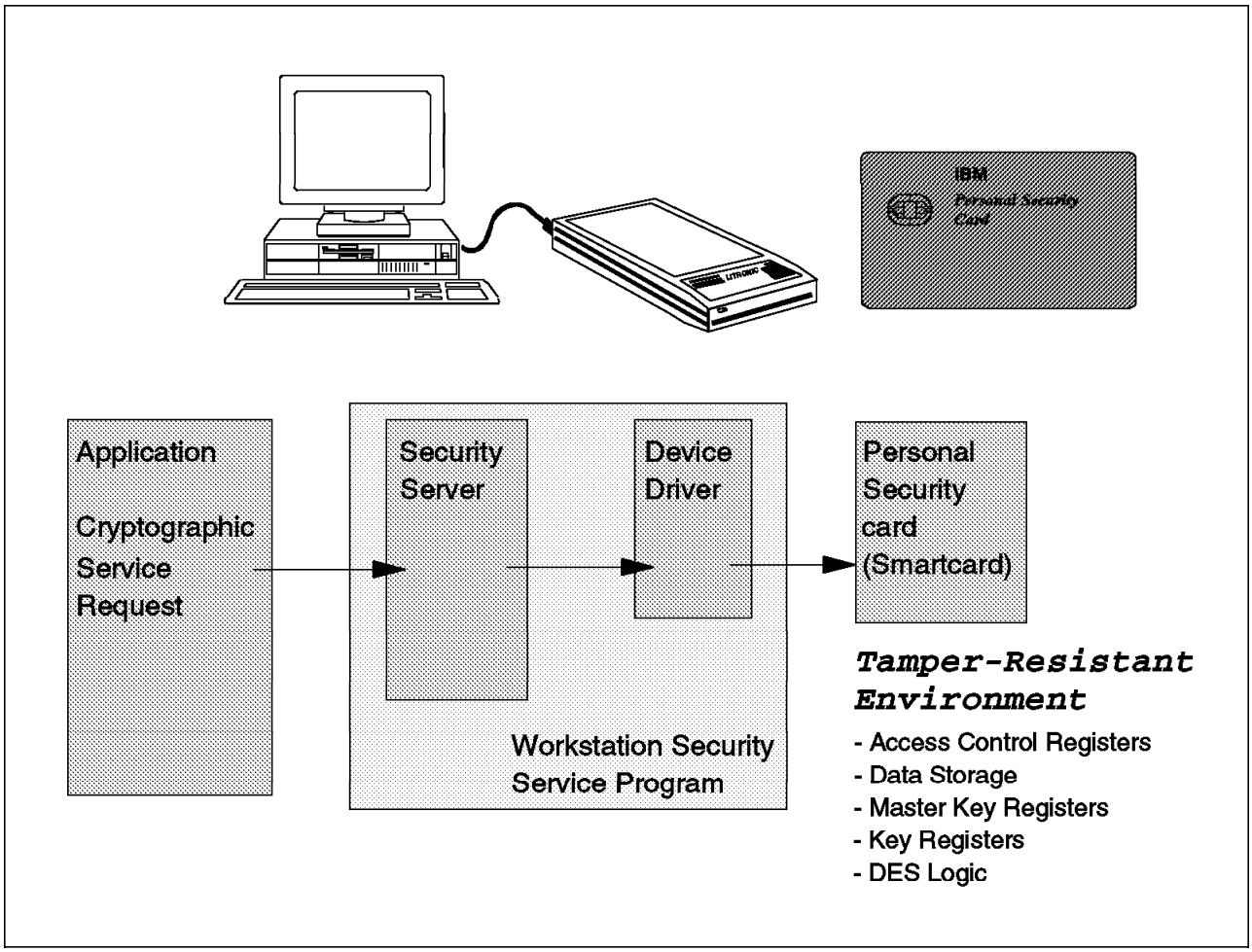

Figure 10. Transaction Security System Server Environment

The cryptographic requests that are issued by application (and utility) programs are delivered to the *security server* component. This server processes the request and usually creates one or more request to the attached hardware.

The security server calls a device driver to request the I/O component to perform hardware commands.

Each I/O component (cryptographic adapter, security interface unit , and Personal Security card) contains a tamper-resistant area that contains the following:

- Command logic
- Access-control logic
- Cryptographic processing logic
- Cryptographic key registers including a master key register

In addition, the security interface unit also contains a clock-calendar; the Personal Security card also contains data storage areas. The tamper-resistant sub-environment permits functions to be performed securely even though the remainder of the computer environment might be subject to attack.

5.6 TSS Software

The TSS Software is provided by different products for the various environments.

- IBM *Workstation Security Service Program*. This program provides an access method and utilities for the workstation environment. The Workstation Security Service Program enables the use of the cryptographic adapter, the security interface unit, and the Personal Security card. This software is designed for use in the DOS and OS/2 environments.
- IBM *AIX Security Service Program/6000*. This program provides an access method and utilities for the RISC System/6000* computers in the AIX environment. The AIX Security Service Program/6000 (AIX/6000*) enables the use of the cryptographic adapter, the security interface unit, and the Personal Security card. This software is designed for use in the AIX client machine environment.

Both software products provide, among many others, the following facilities:

- Device drivers
- Security API
- Hardware Initialization and Key Management (HIKM) utilities

Generally, a installation will install the TSS software in a system and application development area. The installation then can select and customize the components of the TSS software and combine these with the application program (such as the signon application) for use in specific workstations or client machines.

5.6.1 Device Drivers

The TSS software uses drivers to communicate with and to control the Transaction Security System hardware components, such as the 4754 Security Interface Unit and the Personal Security card. The drivers that run in the DOS environment are Terminate and Stay Resident (TSR) programs. When the drivers are loaded, they require various parameters.

The drivers require buffers for various cryptographic operations. These buffers are dynamically allocated or can be pre-allocated at the start of the TSS software. The TSS software provides suitable reduced-function drivers and full-function drivers. An installation can customize the device driver to include

support only for the verbs and key management services that are needed (for example, CSNBPTG) by using the software configuration utility.

5.6.2 Security Application Program Interface

An installation can create application programs that use the TSS products. Application programs and utilities obtain services from the Transaction Security System products by issuing service requests (*verb calls*) to the runtime subsystem of the TSS software and hardware. These requests are in the form of procedure calls that must be programmed according to the rules of the language in which the application is coded. The standard services that are available are collectively described as the *Security Application Program Interface* (SAPI).

As stated above, the security API is the primary application programming interface for accessing services provided by the TSS hardware cryptographic products. The security API extends the IBM Common Cryptographic Architecture (CCA) by providing additional services and additions to the existing CCA services. The SAPI is designed for high-level languages, such as COBOL, PL/1, Pascal, or C, and for low-level languages, such as assembler. It is also designed to enable the installation to use the same verb entry-point names and variables in the DOS, OS/2, AIX for RISC System/6000, MVS, and OS/400 environments.

The way in which application programs and utilities are linked to the API services depends on the computing environment. In the DOS (and MVS) environments, linking applications to the services causes the security API *stub code* to be incorporated into the calling program. In the OS/2 and AIX environments, the operating systems dynamically links application security API requests to the TSS subsystem.

A special verb is designed for the Secured Single Signon solution that is described in this document. A PassTicket generate verb is included in the SAPI.

5.6.3 PassTicket Generate Verb

An application program can access the services provided by the TSS hardware, such as the Personal Security card, by using a procedure call. So, if the signon application needs to perform a PassTicket generate operation, it needs to perform a PassTicket generate verb. This will result in a execution of the PassTicket generate routine on the Personal Security card. The Personal Security card then returns the output from this operation to the signon application.

The procedure call for a callable service (verb) uses the standard syntax of a programming language, including the entry-point name of the verb, the parameters of the verb, and the variables for the parameters.

The entry-point name for the PassTicket generate callable service is CSNBPTG.

All information that is exchanged between the application program and a verb is through variables that the parameters for a verb identify in the procedure call. The parameters for a verb must be simple address, direct-memory pointers to the variables that are exchanged with the verb. The information in each parameter is not always used, but the customer must always declare each parameter that is specified for that verb.

The description of each parameter for each verb begins with the direction in which the data flows; Input, Output, or Input/Output, and the type of associated variable.

The format of a PassTicket generate verb is shown in Figure 11.

```
CSNBPTG
return_code
reason_code
exit_data_length
exit_data
blkname
time
passticket
userid
applid
```

Figure 11. Format of PassTicket Generate Verb

The first four parameters of each verb are the same:

return_code Direction: Output. Specifies an address that points to the place in an application data storage that contains an integer value that expresses the general result of processing the verb.

reason_code Direction: Output. Specifies an address that points to the place in an application data storage that contains an integer value that expresses the result of processing the verb. Different results are assigned to unique reason code values.

exit_data_length Direction: Input/Output. The workstation and client machine environment does not support user exits. Although the verbs do not examine the information in this parameter, this parameter must be declared in the parameter list.

exit_data Same as exit_data_length.

The following parameters are CSNBPTG specific.

blkname Direction: Input. This field contains the address of an eight byte string which in turn holds the name of the application data block on the the SSS Card.

time Direction: Input. This field is a pointer to a four byte field in an application data storage that contain the time value for the PassTicket generator.

passticket Direction: Output. This field contains the address of an eight byte string in the application data storage where the PassTicket is stored after successful completion of the callable service.

userid Direction: Output. This field contains the address of an eight byte string in the application data storage where the user ID is stored after successful completion of the callable service.

applid Direction: Output. This field contains the address of an eight byte string in the application data storage where the application ID is stored after successful completion of the callable service.

Each return code is associated with a reason code that supplies details about the result of verb processing. A successful result can, for example, include return code 0 and reason code 0. A difference between the MACVER result and the token in the application data block on the PassTicket generate call results in a return code 8 and reason code 95. (A PassTicket generate call will result in a MACVER operation on the SSS Card to determine the token for the application data block).

return_code = 8 Indicates that the verb stopped processing. Either an error occurred in the application program or a possible recoverable error occurred in the TSS component.

reason_code = 95 Access to the data is not authorized.

Application programs should base their decisions on the return codes; the reason codes amplify the meaning of the return codes.

Chapter 6. Signon Application

This chapter describes the signon application that was designed and tested during this project. The signon application is designed to run on a DOS or an OS/2 workstation. The principles of this sample application can be used to design signon applications for other platforms, such as a RISC/6000 - AIX client machine.

An overview of the activities to sign a user to a host application is the following:

1. The user inserts the Personal Security card in the smart card reader and enters the PIN from the workstation keyboard or the keypad on the reader.
2. After a successful authentication, a list of host applications for which the user is authorized is presented by the signon application.
3. User selects the application of his choice. This selection results in a call to the PassTicket generate function on the Personal Security card. This function produces a PassTicket for the user to that particular application, and is forwarded through the EHLLAPI interface instead of the password to the server application.

6.1 Loading the Workstation Software

The workstation can be configured in such a way that after a Power-on the various software components get automatically loaded. Among these software components are:

- DOS or OS/2 with a DOS-box
- Communications Manager/2 or another 3270 emulator
- Workstation Security Service Program (including the device driver for the smart card reader)

The signon application can be started from an icon on a utility panel.

6.2 End User Authentication

When the signon application is started, the user is asked to insert his smart card into the reader. The program then issues a CCA ″card-contact″ command to supply power to the Personal Security card and activates the default profile (PSCUSER0).

When activating the profile, the user is asked to enter his PIN. If the PIN is successfully verified, the program proceeds to read the contents of the directory block. If the PIN is incorrect, the user will be requested to re-enter his PIN until he reaches the “Verification Failure Limit” that is specified for this user in his user profile. Once a user reaches his failure limit, he will not be able to use the card until the administrator (with sufficient authority) resets his failure count.

6.3 Presenting List of Authorized Applications

The list of host applications accessible for the user are stored as records in a "directory" data block. This directory data block is a normal Personal Security card data block. The fields of this block are of variable length, and separated from each other by the standard C-language end-of-string character '\0'.

The directory block begins with header information that serves as an index to the rest of the data in the block.

The actual application definitions are located in the data block after the header information. An application definition contains two fields:

1. *Application name*. Application name contains the name of an application in the form that is understandable to the end user. Application name can be freely chosen, for example: "VM/CMS and E-Mail in POK" or "NV/AS Appl. Server in Raleigh."
2. *Application block name*. This field contains the name of the smart card data block which holds the application name, the user ID, and the number of the key register containing the secret application key.

The signon application uses the contents of the directory block to build a list of authorized applications for this user.

From the header information, the signon application determines how many keys and corresponding applications are stored in the Personal Security card. After that, it reads as many pairs of application names and application block names as are defined and constructs a list.

The signon application then presents this list to the end user on the screen, and the user is requested to select an application. The signon application then waits for the user to select an application.

6.4 PassTicket Generation

When the user selects an application from the list, the signon application selects the corresponding table entry from the table of the application block names. It then calls the C-language time function to acquire the time in seconds after the 1st of January 1970, and passes that information along with the application block name to the PassTicket generation function. The call to the PassTicket generation function (CSNBPTG) executes a routine in the Personal Security card. This routine uses the information stored in the application data block and in the key register along with the information passed to it to generate a PassTicket.

An application data block consists of the following:

- The Applid is the application ID of the VTAM* application. The server should use the same application ID. If, for example, the server is located in a MVS - RACF host environment, then the same application ID should be used to define profiles in the PTKTDATA class.
- User ID is the user ID that is known to the server application.
- Key-register number is the number of the key register in the Personal Security card where the secret key for this application resides.

The routine on the Personal Security card that executes the PassTicket generator also checks the cryptographically protected field in the application data block. If the content of a data block or the secret application key is changed, the cryptographic check will fail and the PassTicket will not be generated. The PassTicket generate call will return a nonzero return code to the signon application.

If the cryptographic check is successful, the PassTicket will be generated and sent to the signon application together with the user ID and the application ID that were used to generate this PassTicket. A return code of zero is also sent to the calling signon application.

6.5 Communication with the Server Application

After generating the PassTicket, the signon application is ready to start the communication with the 3270 host applications through the EHLLAPI programming interface of the 3270 emulator used. The input to this step is read from a *logon script file* that contains the actions to be executed to access a VTAM application.

There is a logon script file for every server application the user may access.

6.5.1 Logon Script File

Some of the highlights of the design of this logon script file are:

- In the logon script file, multiple entry points to a VTAM application can be defined. This is achieved with the ′LOOK_FOR:′ constant string in the logon script file, defining the string in the VTAM screen that the program is looking for.
- The contents of the logon script file does not include any security sensitive user dependent information included. User ID and PassTicket information are replaced by the &UID and &PT variables (refer to **A** in Figure 12), that are substituted with the real user ID and PassTicket values coming from the Personal Security card through the PassTicket generate function call. This means that the logon script file can be distributed to the end-user population without the need to tailor them separately for each user.
- All the VTAM application dependent information is stored in the logon script file, for example, no hard-coded application dependant values exist in the signon application code. This means that to add a new VTAM application to the signon application it is only necessary to create a corresponding script file in the workstation. Also, if the access to an application is changed in the host, only the script file has to be changed.

After a PassTicket function call, the signon application has acquired the server application ID from the card. The signon application then searches for a script file named according to a naming convention created for this signon application: the application ID concatenated with the extension name ′SCR′.

Figure 12 depicts the layout of a script file.

```
* Define the host session and the timeout for host screen searches
SESSION: A
TIMEOUT: 30
LOGON APPLID(applid)@E
LOOK_FOR: character_string_1
&UID@E&PT@E   A
```

Figure 12. Format of a Logon Script File

6.5.2 Logon Script File Interpretation

The signon application interprets the logon script file in the example in the following manner. The numbers in this list correspond to the respective lines in Figure 12.

1. Any line which is empty, starts with *, or contains a blank in column 1 is considered to be a comment.
2. The keyword “SESSION: ”starting in column 1, followed by the session ID tells the program which session to use when processing the following statements, until the end of the script, or until another SESSION: statement is executed.
3. The keyword “TIMEOUT: ”starting in column 1, followed by a number gives the number of seconds the program will search each host screen for strings specified with the “LOOK_FOR: ” keyword. The default value is 15 seconds.
4. Any line that is not identified as one of the four types described in this overview will be treated as a command to be sent to the host session. This can be any mix of characters and EHLLAPI function codes, like @E to simulate pressing the Enter key, or @C to simulate pressing the Clear key.

   This allows the logon script file to enter a direct logon to a specified APPLID by executing the following command:

   LOGON APPLID(*applid*)

   followed by the @E character, which represents the Enter key on the 3270 terminal.

   The server application (if available) will respond with the (customized) logon screen for that application.
5. The keyword “LOOK-FOR: ” starting in column 1, followed by the string you wish to find on the screen. The program searches the screen once each second, stopping when the string is found, or when the number of seconds it has been searching reaches the TIMEOUT value. The string can contain multiple words and is case sensitive.

   To make sure that the logon panel for the requested application is displayed on the screen, a unique character string should be specified here. If possible, for example, specify the name of the company or the location where the application is running. This information should of course be available on the logon panel. If the string cannot be found, the program skips through the input lines until it finds the next line containing the ′LOOK_FOR:′ string.
6. If the string was found in the 3270 screen, this line is forwarded to the 3270 session. All the input to one 3270 screen is to be contained in one line, ending with a ′@E′ character, which represents the Enter key. After a line is

processed by EHLLAPI, the program pauses for a short period of time to let the host process the input.

If &UID and &PT are found on a line containing input to a 3270 screen, they are replaced with the information from the Personal Security card. &UID is replaced with the user's user ID for the server application and &PT with the PassTicket, generated for this user for the specific server application he is currently logging on.

6.5.3 Logon Script File Samples

This section describes some samples of logon script files.

To logon to the Time Sharing Option (TSO/E) on the MVS/ESA host, the logon script file needs to contain the following commands. Refer to Figure 14 on page 56 for a sample TSO/E Full Screen Logon Panel.

```
* Define session and timeout values and logon to TSO/E
SESSION: A
TIMEOUT: 10
LOGON APPLID(TSO_applid)@E
LOOK FOR: IKJ56700A ENTER USERID
&UID@E
LOOK FOR: Enter LOGON parameters below:
&PT@E
```

To logon to a CICS application on a MVS/ESA host, the logon script file needs to contain the following commands. Refer to Figure 13 on page 54 for a sample CICS welcome screen.

```
* Define session and timeout values and logon to CICS
SESSION: A
TIMEOUT: 12
LOGON APPLID(CICS_applid)@E
LOOK FOR: DFHCE3520 Please type your userid
&UID@E&PT@E
```

If a session manager, such as NetView Access Services, is installed, the signon application can be used to logon to this session manager. The signon application can also be used to utilize the *automatic logon* facility of this session manager. Refer to 7.4.6, "Automatic Logon and Logoff" on page 64 for a short introduction of this automatic log on facility in NV/AS.

Once the signon to the session manager is established, the signon application can display the Application Selection panel from NV/AS. See Figure 18 on page 62 for a sample of the NV/AS Application Selection panel. In order to utilize the automatic logon facility, the signon application can now enter an ID as specified on the Application Selection panel. This will start the automatic logon by the session manager. This logon will also use PassTickets to logon to the requested applications.

The following commands should be specified in the logon script file to signon the end user to OpenEdition* MVS on TSO/E in Poughkeepsie. The recorded logon file within NV/AS should contain the input parameters for the TSO/E LOGON panel; after entering the &UID and &PT, the OMVS command should be entered. These commands are entered by the end user when he performs his first signon to OpenEdition MVS through the session manager NV/AS. Refer also to Figure 18 on page 62 and Figure 14 on page 56.

```
* Define session and timeout values and logon to NVAS
SESSION: A
TIMEOUT: 11
LOGON APPLID(NV/AS_applid)@E
LOOK FOR: NetView Access Service
&UID@E&PT@E
LOOK FOR: Application Selection
* The user should have a recorded logon file available within NVAS
2@E
```

By using a session manager that is capable of performing a secured signon to the various server applications, the administration on the smart card and the logon script file can be minimized. The user authorization can be performed in the session managers environment. There is no need to describe the various server applications on the workstations or client machines.

For a detailed description of this process, refer to: *Enhanced Exploitation of RACF 1.9.2 - Secured Single Signon Using NV/AS*.

6.6 Establish a Session

The signon application establishes a connection to a 3270 session, and through EHLLAPI simulates the user logon through the 3270 screens to the target application.

Currently the signon application as described in this document expects the predetermined flow of 3270 screens from the entry screen to the end of logon, and has no functions to recover from an uncommon situation caused by VTAM, or by the server application itself.

The logic could be improved by adding code to search for a specific string from the screen after every input. If the string was not found, some error analysis or recovery procedures could be executed.

After entering the input for a 3270 screen, the program pauses for a short period and then continues with the next 3270 screen entry. It does not query the result of its previous input, but expects that the entry was successfully processed.

When the screen to enter user's password is reached, the program enters the PassTicket value to the password field and waits for the user to be logged on with a PassTicket.

Chapter 7. Host Software Configuration

The Secured Single Signon solution described in this document is designed to enable a end user to signon to a variety of host (client/server) applications. Some sample applications are:

- CICS and IMS transactions (application programs)
- TSO/E sessions
- VM/CMS advanced program-to-program communication applications
- The OpenEdition shell and POSIX** applications
- Native VTAM applications
- APPC/MVS applications

The use of a session manager, such as NetView Access Services, simplifies the signon procedure to the various host applications.

This chapter provides an overview of the host software that was involved in the design and test of the Secured Single Signon solution described in this document. This chapter also provides a description of the various logon procedures to these host applications.

7.1 CICS/ESA

CICS is one of IBM′s Online Transaction Processing (OLTP) systems. CICS is available on several platforms, including MVS, VSE, VM, AS/400*, OS/2, AIX, and other vendor′s systems. The MVS version is CICS/ESA.

CICS/ESA is present in almost every “production” MVS installation and is the subsystem used by most end users connected to MVS. A typical CICS end user might be a clerk responding to a customer′s telephone inquiry about his account status. CICS *transactions* (application programs) would be used to display the account status, recent changes, and so forth.

CICS application programmers produce fixed-purpose application programs (“transaction programs”) for CICS systems. The account query example just mentioned is typical. CICS users can use only these fixed-purpose application programs. CICS is not a general-purpose time sharing system. Program development, for instance, is not normally done under CICS.

7.1.1 Identifying CICS Terminal Users

If your are using CICS with RACF security checking, you control access to CICS resources, by CICS terminal users, through the levels of authorization you define in the appropriate RACF-managed resource profiles. You define these authorizations for specific users by adding individual RACF user IDs (or RACF group IDs) to the resource access lists. In CICS/ESA Version 3 Release 2.1, you must use an external security manager, such as RACF, for authorizing access to CICS resources. The CICS internal security mechanisms available in earlier releases are obsolete and no longer supported.

7.1.2 CICS Signon Security

When users signon to CICS through VTAM, but do not intend to authenticate themselves to CICS, they can use only those transactions that the CICS default user is permitted to use. As these are likely to be strictly limited, users must signon and authenticate themselves to obtain authorization to run the transactions that they are permitted use.

By default, users signing on to CICS see the CICS “welcome” or “good morning” panel displayed. The panel is displayed by the CICS-supplied transaction, CSGM. Instead of using CSGM, installations can specify an installation-provided good-morning transaction by using the CICS GMTRAN system initialization parameter. This results in users being presented with an installation defined “welcome” panel.

If users clear the welcome message and enter the CICS signon transaction ID, CESN, or specify CESN as the welcome transaction, CICS displays the *Sign-on panel* shown in Figure 13.

```
                        CICS Sign-on


Type your userid and password:

         Userid   ==>
         Password ==>
         Language ==>



     New Password ==>
     New Password ==>









 PF 3=End
 DFHCE3520 Please type your userid
```

Figure 13. The CICS Sign-on Panel

When users are signing on to CICS, there are several phases to the sign on process.

In the first phase, after the Sign-on panel is completed and sent, CICS calls RACF to verify that a valid user ID with the correct password or PassTicket have been entered, and to determine whether the password has expired in case a password is used to authenticate the user.

During the next phase, once CICS has authenticated the identity of the user by the user ID and password (or PassTicket) check, other authorization checks are performed. In this phase, RACF determines whether the user is authorized to access the VTAM application (CICS region) to which the user is trying to signon.

RACF does this by checking the access list of the CICS application profile defined in the RACF APPL resource class. RACF can also be used to verify that the user is authorized to signon to CICS from that specific terminal by using the terminal ID (termid). These checks restrict the user to signing on only to those CICS regions for which he is authorized, and only from one of the terminals the user is authorized to use. The use of defined terminals can also be restricted to certain times of the day, and to certain days of the week.

7.2 TSO/E

Time Sharing Option (TSO) is the full-function time-sharing subsystem of MVS. It allows users to interactively share computer time and resources. TSO/E is an extension of TSO, containing all the base TSO functions and many enhancements.

TSO/E is used for system administration, RACF administration, and general program development. A TSO user potentially has access to most MVS facilities, unlike a CICS user who is restricted to preprogrammed CICS applications.

Interactive System Productivity Facility/Program Development Facility (ISPF/PDF) provides a panel-based user interface to TSO/E.

Some applications are designed to operate directly under TSO. In such cases, application users must log onto TSO and then start their application. Often, applications use the “dialog” or “panels” interfaces provided by the ISPF menus.

7.2.1 TSO/E Logon

Usually, the term “log onto MVS” is used instead of “log onto TSO/E.” A user cannot directly log onto MVS, he must log onto an MVS subsystem (such as TSO/E, CICS, IMS), or connect to a VTAM application which is, in a loose sense, an MVS subsystem. The TSO/E subsystem is a native part of MVS and the two terms are often used interchangeably.

The TSO/E full-screen logon panel, shown in Figure 14 makes the logon process easier by:

- Saving user attributes from one session to the next
- Allowing program function keys to be used during the logon
- Allowing users to enter commands during logon
- Explaining the error when incorrect information is specified

7.3 OpenEdition MVS

The Portable Operating System Interface (POSIX) is a set of standards being defined by the Institute of Electrical and Electronics Engineers (IEEE) under direction of the American National Standards Institute (ANSI). These standards are based largely on UNIX and derivative systems developed by the University of California at Berkeley and others.

POSIX standards have been embraced by nearly all major computer manufactures, hundreds of software vendors, and information system users.

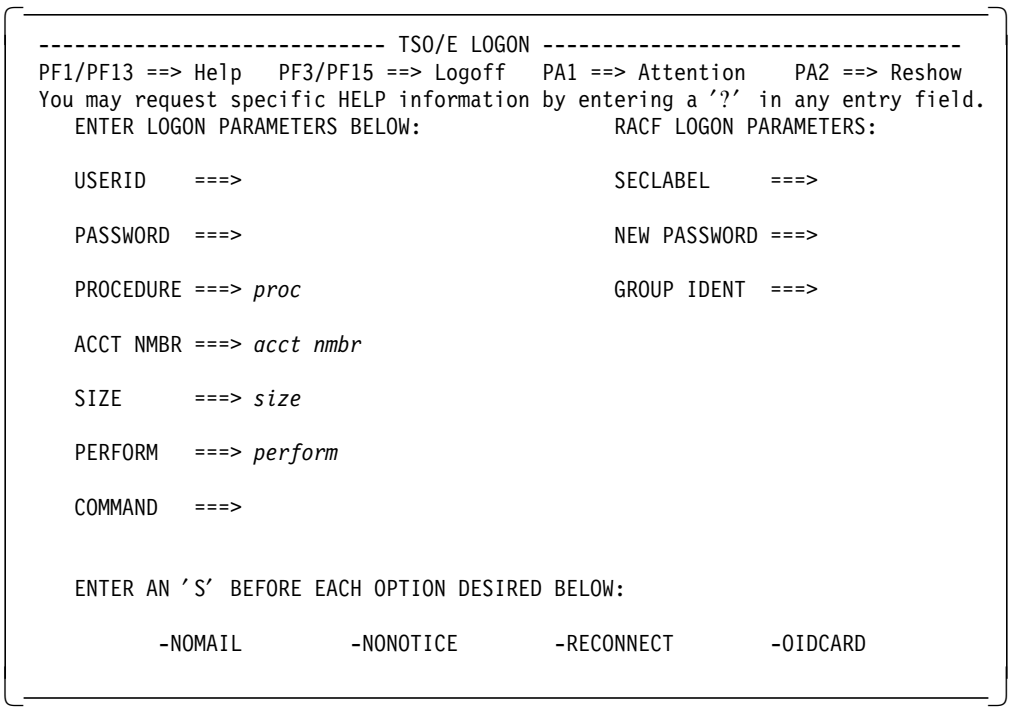

Figure 14. TSO/E Full Screen Logon Panel

The open world promises the economies of standards:

- Strategic skills and knowledge investment
- Portability of applications and data in a multivendor network
- The ability of users and applications to interoperate among systems from different manufactures with a single common interface
- The ability to compare manufactures′ offerings more carefully, to distinguish value-added functions from standard functions

OpenEdition MVS brings together the open world and the MVS world. With MVS/ESA Version 5 Release 1, OpenEdition MVS features are now an integrated part of MVS.

The MVS support for the OpenEdition services component (OpenMVS) allows two open system interfaces on the MVS operating system: an application program interface (API) and, optionally, an interactive shell interface. With the API, the programs can run in any MVS environment - including in batch jobs, in jobs submitted by TSO/E interactive users, and in started tasks - or in any other MVS application task.

The optional shell interface is an execution environment analogous to TSO/E, with a programming language of shell commands analogous to the REXX Language.

Application developers and interactive users using these interfaces have the underlying resources and power of the MVS system available without the need to understand MVS′s own proprietary interface. On the other hand, these users can exploit the MVS proprietary interface for capabilities not provided by standard interfaces.

The interactive user can quickly toggle from the POSIX shell environment to the TSO/E environment to request services not available in the shell. Users can use both OpenEdition MVS services and traditional MVS services, getting the best of both worlds.

Refer to *Introducing OpenEdition MVS* for a more comprehensive overview of OpenEdition MVS.

7.3.1 OpenEdition Shell Session

The OpenEdition shell is modeled after the UNIX System V shell with some of the features found in the KornShell. As implemented for OpenEdition MVS, this shell conforms to POSIX standard 1003.2, which has been adapted as ISO/IEC International Standard 9945-2: 1992.

The shell is a command processor that can be used to:

- Invoke shell commands or utilities that request services from the system
- Write shell scripts using the shell programming language
- Run shell scripts and C-Language programs interactively (in the foreground), or in the background
- Run REXX EXECs

To begin a shell session, you first logon to TSO/E and then invoke the shell with the TSO/E OMVS command. The shell and all processes and process groups running under it are typically in the same session. The user can start multiple shell sessions simultaneously when he logs into the shell with a parameter to the OMVS command, and he can start additional shell sessions at any time during a shell session. The user can switch from session to session, using a function key or subcommand.

For more information on how to use OpenEdition MVS, see the *MVS/ESA: OpenEdition User's Guide*

7.3.2 Hierarchical File System

OpenEdition files are organized in a hierarchy, as in a UNIX system. All files are members of a *directory*, and each directory is in turn a member of another directory at a higher level in the hierarchy. The highest level of the hierarchy is the *root directory*.

7.3.3 OpenEdition Security

If RACF is available, the security administrator can define for an OpenEdition MVS user an *OMVS user ID (UID) and group ID (GID)*, numeric values associated with a TSO/E user ID. The UID and GID are included in the RACF user and group profile when a user is authorized to use OpenEdition MVS. The system uses the UID and GID to identify the files and processes that a user has access to.

The user can control read, write, and execute access to his files by other users in his group or outside his group by setting the permission bits associated with the files.

Typical UNIX security procedures differ from typical MVS security procedures. For an complete overview of the security in an OpenEdition MVS environment see the following publications:

- *MVS/ESA: Planning: OpenEdition MVS*
- *RACF Security Administrator's Guide*

7.4 NetView Access Services

NetView Access Services (NV/AS) is a VTAM application program that runs on the host in an MVS or VM environment. NV/AS is designed to make it easier for users of 3270-type terminals in an SNA network to gain access to applications while ensuring data security in the network. The user can gain concurrent access to several host applications, and easily move between them.

Host applications handled by NetView Access Services can be systems such as Customer Information Control System (CICS) or Information Management System (IMS), subsystems such as Professional Office System (PROFS), or transactions in a subsystem. NV/AS uses VTAM to communicate with applications. NV/AS is a member of the NetView* family of products.

NetView Access Services provides the following functions for a 3270-type terminal user:

- Simplified application selection.
- Multiple concurrent application sessions.
- Application status indication.
- Automatic logon to, and logoff from, applications.

  When users logon to and logoff from any application for the first time, they can record profiles that NetView Access Services uses to logon and logoff from the application on their behalf. This avoids the time spent in repeatedly typing in the same logon and logoff sequences.

  If a specific application is used more than others, NetView Access Services can be made to automatically logon to this application whenever the user logs on to NetView Access Services.
- Disconnects and keeps sessions active.
- Copies data from one application to another.
- Prints a panel.
- Shared sessions with another user.
- Multiterminal sessions.
- Dynamic multilingual support.
- Broadcast and application message indication.

For further information, refer to *NV/AS User's Guide*.

Figure 15 provides an overview of the NetView Access Services functions.

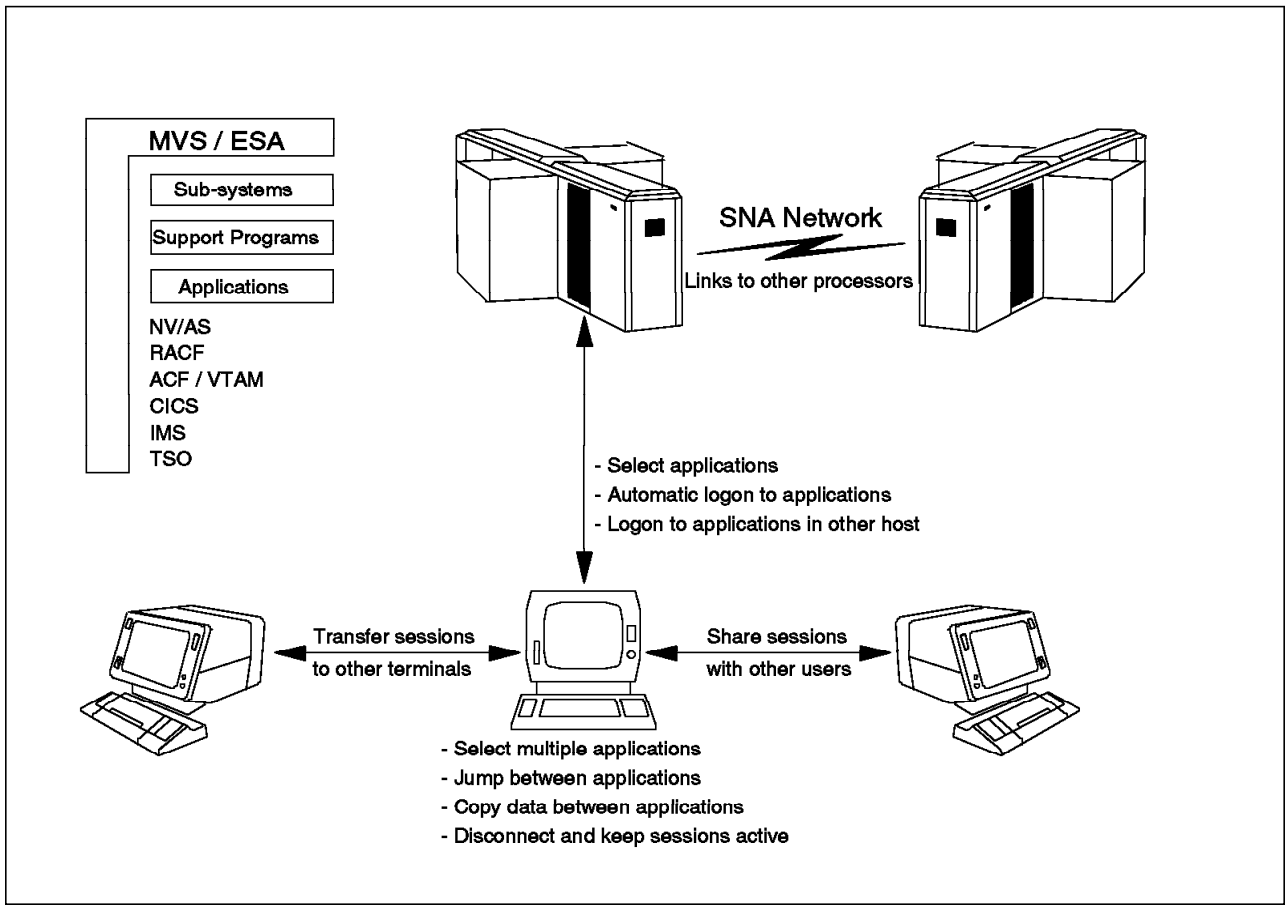

Figure 15. Main User Functions of NetView Access Services

The following functions secure the network and applications against unauthorized use:

- Single access point for all host applications
- User access administration
- Additional user exits for logon authorization and session initialization
- Security time-outs

7.4.1 NetView Access Services Administration

NetView Access Services provides services for the following types of administrators.

7.4.1.1 NV/AS System Administrator

A system administrator does general administration for the entire NetView Access Services system. The system administrator's tasks include:

- Defining each application to NetView Access Services
- Deciding how many and what types of groups there will be
- Assigning the applications to be used by each group
- Defining one or more administrators for each group

7.4.1.2 NV/AS Group Administrators

One or more group administrators are responsible for all information necessary for running the group. The group administrator's tasks include:

- Defining each user to the group
- Setting up access parameters for the applications the users are authorized to use

NetView Access Services administrators carry out their tasks by using panels that can be selected from the Administration Selection panel. To get this panel, log on to NetView Access Services and enter the ADM command on the Application Selection panel. Figure 16 shows the complete Administration Selection Panel in more detail. To select an option, enter the number of the option in the Command line.

```
                    Administration Selection
                                                              Terminal:

Select one of the following:


          1  -  Maintain User Parameters

          2  -  Maintain Group Parameters
          3  -  Assign a User to a Group

          4  -  Define an Application for the System
          5  -  Define a Group
          6  -  Assign an Application to a Group
          7  -  Define an Administrator
          8  -  Broadcast a Message
          9  -  Broadcast Selectively
         10  -  Define BIND User Data for Applications
         11  -  Refresh Profiles from VSAM Data Sets



Command ==> ______
PF 1=Help  3=End
```

Figure 16. Administration Selection Panel

Option 1 is an end-user function. You can use this, for example, to define the parameters for the recorded-logon facility.

Option 2 is a group administrator task. Using this function, you can display and update the parameters of an application that the group has access to. With this function, you can specify whether the members of this group can use automatic logon for this application. Also, you can specify the default logon profile for this application, for all members of this group.

Option 3 is also a group administrator task. Using this function, you can define which applications each member of the group can use, certain parameters for each application, and certain authorizations for the member. With this function, for example, you can specify whether a user can record a logon for this application.

Option 4 is a system administrator task. Before an application can be used by any user of NetView Access Services, it must be defined to the NetView Access

Services system. Using this function, you can select whether logon to this application can be automated by using a recorded logon.

For group administrators, only the first three options are visible on the Administration Selection panel. For system administrators, all options are displayed. Users that are not administrators do not see this panel at all. They see the Maintain User Parameter panel when they use the ADM command.

Refer to *NV/AS Administration* and *NV/AS User's Guide* for more detail on the administration tasks.

7.4.2 Logging On to NetView Access Services

Before you can logon to NetView Access Services and gain access to your applications, your authorization must be checked. This means you need a user ID and password or PassTicket You also need to have been defined to a group and assigned applications in that group.

If you logon to NetView Access Services by using, for example, a logon command such as:

LOGON APPLID(*NVAS_applid*)

you will get the NetView Access Services Logon Panel on your 3270-screen. The layout of this panel is shown in Figure 17. The layout of this panel may differ from location to location.

```
                                                          Hot Line: 293-1660
                                                          Data . .: 07/26/94
                                                          Time . .: 19:01:28
                                                          Terminal: SCGSC042
       N E T V I E W
       ********   *******    *******    *********  ******       *******
      **     **  **     **  **     **  **         **   **      **    **
     **     **  **         **         **         **           **
    *********  **         **         *********    *******      ********
   **     **  **         **         **                 **            **
  **     **  **    **   **     **  **           **     **    **      **
 **     **   ******      ******   *********     *******      *******

      NetView Access Services, Release 3.1 - Program Number 5665-365
        (C) Copyright IBM Corp. 1987, 1991.  All rights reserved.

             Enter Logon informatie:
               User . . . . . .              (User ID/LOGOFF)
               Password . . . .              New Password . .

               Application  . .
               Group  . . . . .
               Location . . . .

 PF 1=Help  2=Language
```

Figure 17. NV/AS Logon Panel

For a complete description of the possible replies on this panel, refer to the *NV/AS User's Guide* .

7.4.3 Selecting an Application

When you have logged on, NetView Access Services displays an Application Selection panel. It should be similar to the panel shown in Figure 18 and lists the applications that you are authorized to use. Applications that have the time in the Status field highlighted are available for use.

```
                          Application Selection    Help: 293-1660 Term: SCGSC042
                                                   Data: 07/26/94 Time: 19:01:28
                                                   Broadcast:
Select application or enter command. Return to this panel using Escape key PA2
Issue commands in applications using Command key PF10 and prefix $$

  ID  Name     Status M  B Jump Key Application Description
   1  WTSCPOK  13:59                VM/CMS and E-Mail in POK
   2  WTSCMXA  13:59                TSO/E in Poughkeepsie
   3  NVASR01  13:59                NV/AS Appl. Server Raleigh
   4  WTSCNET  13:59                Network Machine
   5  CICSA    13:59                CICS in Poughkeepsie
   6  IMS410   13:59                IMS in Raleigh
   7  RALYDPD6 13:59                VM/CMS and E-Mail in Raleigh




To terminate all sessions use the LOGOFF command.


Command ==> ____________________  ________  ________  ________
PF 1=Help  2=Language   3=Disconnect   4=Redefine Keys   7=Backward   8=Forward
```

Figure 18. Application Selection Panel

To select an application, do one of the following:

- Position the cursor to the left of the application ID and press Enter.
- Type the name or the ID of the application in the command line and press Enter.

NetView Access Services responds to your selection by displaying the first panel of the application, which is usually a logon panel.

There are different ways of selecting an application. For example, NetView Access Services provide a facility, called “Recorded Logon” that provides an automatic logon and logoff facility. Refer to Section 7.4.6, “Automatic Logon and Logoff” on page 64 for more detail. See also *NV/AS User's Guide* for other application selecting procedures.

7.4.4 Modes of Access

NetView Access Services offers two types of access a user can have to an application. These modes are *relay mode* and *pass mode*.

7.4.4.1 Relay Mode

In relay mode, all communication with an application is controlled by NetView Access Services. All the NetView Access Services functions such as hot-keying among applications, automatic logon, and copying data from one application to another are available to the user.

7.4.4.2 Pass Mode

Using an application in pass mode bypasses NetView Access Services and, therefore, all the relay functions. This mode can be used when a user only needs to use a particular application or to save system resources.

Whether an application can be used in pass mode, relay mode, or both is defined by the system and group administrators.

7.4.5 Customizing NetView Access Services

Installation exit routines are called by NetView Access Services at strategic processing points or exit points. Each installation exit handles a particular event such as, in the example of user authorization checking, the receipt of a user logon request.

When that event occurs, NetView Access Services passes control to the appropriate installation exit routine for processing. After processing, the installation exit routine returns control and passes a return code to NetView Access Services. Sample exit routines are provided by NetView Access Services. These exit routines can be tailored to the need of the installation.

7.4.5.1 Logon Authorization Exit (EMSELGNX)

NetView Access Services provides an internal and an external security interface. Using internal security, the users can be identified by NetView Access Services through a valid user ID that must be recorded in the NV/AS database.

Since customers have requested IBM to provide a single user registry and a consistent method of user registration and authentication with a common logging and auditing facility, NetView Access Services provides an external security interface. In the MVS and VM environment, this requirement is satisfied by using RACF (or any other external security package) as much as possible.

The philosophy behind the Logon Authorization Exit is as follows:

- If the installation policy says that a valid user ID is sufficient to get access to the session manager, you should use the standard NetView Access Services code. Access control to applications can still be (more restrictively) controlled by an external security interface.
- If the installation policy says that a valid user ID and password (or PassTicket) is required to identify and authenticate the user in the session manager, you can use the standard shipped and maintained Logon Authorization Exit (EMSELGNX). Installations can tailor this installation exit, for example, to interface with a product other than RACF, or implement even a “secondary authentication” routine.

Note: The standard Logon Authorization Exit routine provides a System Authorization Facility (SAF) interface, by using RACROUTE REQUEST=VERIFY, instead of the RACF interface RACINIT. The term “RACINIT” originally referred to the independent RACF system macro. The RACINIT macro was replaced by RACROUTE REQUEST=VERIFY or RACROUTE REQUEST=VERIFYX macro.

Since many customers are familiar with the term RACINIT, this term is still used in some documentation.

The exit routine EMSELGNX is invoked when user ID and password information is entered on the NetView Access Services Logon panel or when BIND user data is sent from the application that calls NetView Access Services (for example VTAM). If a nonzero return code is returned, NetView Access Services resends the Logon panel with a message, depending on the return code.

7.4.6 Automatic Logon and Logoff

When NetView Access Services does automatic logon or logoff for an end user, it calls a logon profile for the requested application. An automatic logon profile is a file that consists of the interaction between the terminal and the application for the logon and logoff processes. It consists of a logon sequence and, if recorded, one or more alternative sequences, and a logoff sequence.

There are three kinds of logon profiles:

- Those defined by the system administrator for all users on the NetView Access Services system
- Those defined by the group administrator for all members of that specific group
- Those defined by the end user

A logon sequence consists of several interactions between the terminal and an application necessary to logon to an application. If such a sequence of logon data (for example user ID and password) is recorded, it is used to generate the same logon later automatically.

When the end user records an automatic logon profile, he can:

- Enter all the normal logon data as entered on the application log-on panel
- Enter variables as well as the normal logon data as entered on the application log-on panel

For example, instead of a user ID and password, a user could enter the variables &UID and &PWD. These variables are stored with the rest of the logon input and are assigned values before or during automatic logon. The &PWD value is retrieved from a NetView Access Services control file where it is masked with a proprietary masking algorithm. The &UID and &PWD variables are called ampersand variables.

Notes:

1. The entire recorded logon data stream is stored within a NetView Access Services control file. If the user is not using ampersand variables, there is no way to identify the various fields in this data stream, but there is still a user ID and password in clear text in this data. For very obvious security reasons, end users should not use this way of recording.
2. The implementation of the Secured Single Signon facility, as described in *Enhanced Exploitation of RACF 1.9.2 - Secured Single Signon Using NV/AS*, requires that the user record his logon using ampersand variables.

   Since the ampersand variables have identifiers in the recorded logon data stream, they can be checked and, in case of the Secured Single Signon

facility, be replaced by values that are generated in the Secured Single Signon facility.

In the samples in the above mentioned document, the &PWD is replaced by a PassTicket that is generated for this session.

Refer to Section 7.4.7, “NetView Access Services Users Functions” for a description of how to enter these variables.

7.4.7 NetView Access Services Users Functions

Among many other functions, NetView Access Services lets an end user customize NetView Access Services to his own requirements. These personal administration functions enables an end user to assign values to variables that can be used as part of an automatic logon profile for specific applications. These variables are called “User Parameters” or “User Application Access Parameters.”

On the NetView Access Services Maintain User Parameter panel, the end user can specify the user ID that must be used to logon to this specific application. This user ID does not have to be the same user ID that was used to logon to NetView Access Services. Refer to Figure 19 for the panel layout.

```
                          Maintain User Parameters
                                                          Terminal:
Application . . .________                                 User . .:
Last Update . . :                                         Group. .:
                                                          Default Group:
Fill in or change the following:
  Selection ID . . . . .  __   (1-99)      Default Application _ (Y/N)
  Msg. Received Indicator _    (N=Normal/J=Jump/I=Information)
  Jump Key . . . . . . .  ___  (PFnn/PAnn/ATTN)
  Logon Profile  . . . .  _    (U=User/G=Group/S=System)  2
  Active Profile . . . .  _    (1/2) user profile
  Profile 1 Comment. . .  ____________   Profile 2 Comment ____________
  Application ID Display  -    (Y=Yes/N=No)

Variables for Logon profile:

  &UID . . . ________      &UVAR1 . . ________      &UVAR2 . . ________
  &ACCNO . . ________      &UVAR3 . . ________      &UVAR4 . . ________
  &PWD . . .               &NPW . . .               &OPW . . .

Enter a command: d (display), u (update, or l (list).


Command ==> ______
PF 1=Help  3=End
```

Figure 19. Maintain User Parameters Panel

The usage of the recorded-logon facility can be authorized by the various NetView Access Services administrators. The authorization can be specified as system-wide, to a group of end users, or for each application for each member of a group of end users.

Note: Although there are certain restrictions, this recorded logon facility introduced a security trap when implementing the Secured Single Signon facility. The specification in the recorded logon of a different user ID for a requested

application than the user ID that was used to logon to NetView Access Services, could easily be allowed in normal NetView Access Services processing since the user ID and password combination are checked at the moment the user is logging on to the application.

The current implementation of Secured Single Signon generates a PassTicket for every end user who is validated by this NetView Access Services regardless of the user ID that is specified for the requested application. An additional security check is implemented in NetView Access Services Session Establishment Exit to verify the authorization for this specific user ID for the requested application. This security check is based on:

- The ACB name of the requested application (target application)
- The ACB name of the requesting NetView Access Services
- The requestor's current user ID (used to log on to NetView Access Services)
- The target application user ID

7.5 Resource Access Control Facility

Resource Access Control Facility (RACF) for MVS is a program product that works together with the existing system features of MVS/SP*, MVS/XA*, or MVS/ESA to provide improved data security for an installation. RACF for VM works with VM/Enterprise Systems Architecture (VM/ESA*).

RACF helps to improve the data security by providing the ability to:

- Identify and verify users
- Authorize users to access the protected resources
- Control the means of access to resources
- Log and report various attempts of unauthorized access to protected resources
- Administer security to meet an installation's security goals.

RACF provides these functions; the installation defines the users and the resources to be protected.

A specific RACF user, called the security administrator, has the responsibility to define users and resources to RACF. As well as defining what resources to protect (DASD data sets, minidisks, terminals, CICS or IMS transactions, application signon, and so on), the security administrator can define and grant the authorities by which users access the protected resources. Alternatively, the security administrator can assign other people to do some of this defining.

RACF retains information about users, resources, and access authorities in profiles on the RACF database and refers to the profiles when deciding which users should be permitted access to protected system resources.

7.6 Identifying and Verifying Users

RACF uses a user ID to identify the person who is trying to gain access to the system and a password to verify the authenticity of that identity. RACF uses the concept of only one person knowing a particular user ID - password combination to verify user identities and to ensure personal accountability.

Having identified and verified the user, RACF then controls interaction between the user and the system resources. RACF must authorize not only which users may access resources but also in what way the user may access them, such as for reading or for updating. RACF also can authorize when a user can access resources, by either time or day.

7.7 RACF Secured Signon

The RACF Secured Signon function provides an alternative to the RACF password associated with specific user profiles. It is called a *PassTicket* and allows workstations and client machines in a client/server environment to communicate with the host without using a RACF password.

The RACF PassTicket removes the need to send RACF passwords across the network and allows a function other than RACF to authenticate a mainframe application user ID.

It also allows an installation to move the user authentication part of the host signon function from RACF to another product or function. This gives that product or function the ability to allow or deny access to host applications.

Notes:

1. If your installation takes advantage of the RACF Secured Signon capabilities and allows a function or product other than RACF to authenticate users, that function or product must assume the responsibility of ensuring the security of the user authentication checking function.
2. It is the responsibility of the function that generates PassTickets to ensure that an end user can only generate PassTickets for applications and RACF user IDs to which that user is authorized by RACF on the host.

The RACF Secured Signon function can be used on MVS and VM systems with the CICS, IMS, TSO, and VM/CP signon protocols without making changes to them. It also works with MVS APPC signon and MVS batch jobs. Every other host application can use the RACF Secured Signon function through a default secret key. After some small changes in the user validation process, every host application can use its own specific secret key to validate PassTickets.

The RACF PassTicket generator algorithm can be installed on many other platforms besides these that are capable of running MVS or VM. The algorithm is published and can, for example, be installed in a DOS, OS/2, or AIX environment.

7.7.1 PassTicket Algorithm Input

To successfully use the PassTicket, the target application, using RACF to identify and authenticate a user ID, must have specific input information for the algorithm. These inputs are:

- *The RACF user ID for the target application*. The user ID is passed to RACF by the application during the RACINIT call that is performed by the application.
- *The application name of the target application*. This name is used to determine the profile name in the PTKTDATA class. This profile name is used to describe this application. Refer to Sections 7.7.3, “Defining

PTKTDATA Class Profiles" and 7.7.4, "Determining Profile Names in the PTKTDATA Class" for a description of this profile.

- *The RACF Secured Signon application key*. RACF will, internally, retrieve the secret application key by using the profile in the PTKTDATA class.
- *Time and date information*. To limit the usability of a PassTicket to a number of seconds, the PassTicket generator needs a time stamp.

  The input data must be a 4-byte binary number that shows how many seconds have elapsed since January 1, 1970, at 00:00 Greenwich Mean Time (GMT). Refer to Section 9.1, "Host Time" on page 77 for more detail on this topic.

The RACF PassTicket generator algorithm uses the input information to create a PassTicket. The information passes through the algorithm, which uses a cryptographic technique, to ensure that each PassTicket is unpredictable.

The PassTicket is an 8-character alphanumeric string that can contain the characters A through Z and 0 through 9.

For a complete description of the PassTicket algorithm, refer to *Resource Access Control Facility: Macros and Interfaces*.

7.7.2 Activating the RACF Secured Signon

Before using the RACF Secured Signon function, you must activate the RACF class PTKTDATA. This class contains all the profiles with the PassTicket information. To activate the class and the RACF Secured Signon function, issue the following RACF commands:

```
SETROPTS CLASSACT(PTKTDATA)
SETROPTS RACLIST(PTKTDATA)
```

When you have activated the PTKTDATA class, you can define the necessary profiles.

7.7.3 Defining PTKTDATA Class Profiles

You must create a profile in the PTKTDATA class to identify each application that users are allowed to access with the PassTicket. The purpose of the profiles is to associate a secret RACF Secured Signon key with a particular application on a particular system. To define the profile, use the RDEFINE command. For example, enter the following command:

```
RDEFINE PTKTDATA profile_name
    SSIGNON(KEYMASKED(key_value))
    UACC(NONE)
```

The variable *key_value* represents 16 hexadecimal characters providing an 8-byte key. The value of that key must be specified as a combination of the following letters and characters:

```
A B C D E F 0 1 2 3 4 5 6 7 8 9
```

7.7.4 Determining Profile Names in the PTKTDATA Class

Depending on the application, the RACF Secured Signon function uses a specific method for determining profile names in the PTKTDATA class.

The RACF PassTicket validation algorithm uses the application name that is passed in the RACINIT performed by the specific application to identify and validate the user. A typical example of such user validation might look like:

```
RACROUTE REQUEST=VERIFY ENVIR=CREATE,USERID=userid       X
    PASSWORD=password,PASSCHK=YES                        X
    APPL=applname,ACEE=acee_address
```

If a PassTicket is generated for this user ID, it is placed in the PASSWORD field in this RACROUTE call. The PassTicket validation function will use the *applname* in the APPL parameter to search for the profile in the PTKTDATA class. This profile contains the secret RACF Secured Signon key for this application. Host applications, such as CICS, IMS, and APPC, use the standard naming conventions to define these applications to the APPL class.

Other applications, such as TSO and MVS batch jobs, do not use any *applname* during the RACINIT processing. To allow these applications to use the RACF Secured Signon functions, the following standard is defined:

- Create the profile name to represent the TSO application to the PTKTDATA class by prefacing the SMF identifier of the system with the characters **TSO**.

  The SMF identifier for this specific system can be found in the SMFPRMxx member of SYS1.PARMLIB and specified by the SID keyword. Check the IEASYSxx member of SYS1.PARMLIB you use to IPL for the keyword SMF to see whether your are using the correct SMFPRMxx member.

  If more TSO systems are using the same RACF database, then the systems must be set up with different SMF-IDs. If, for example, the SMF identifier of the system on which the TSO application is running is SYS2, the name of the profile should be: TSOSYS2.

- MVS batch jobs that include RACF passwords in the JCL can replace the password with a PassTicket

  Create the profile name to represent the MVS batch job application to the PTKTDATA class by prefacing the SMF identifier of the system with the characters **MVS**.

  Use the same procedure as described for the TSO application to find the SMF identifier. If, for example, the SMF identifier of the system on which the MVS batch job is running is SYS2, the name of the profile should be: MVSSYS2.

For a more detailed description of this naming standard for the PTKTDATA class profiles, refer to *Resource Access Control Facility - Security Administrator's Guide* and *Resource Access Control Facility - Macros and Interfaces*.

Note: Because each application must be uniquely defined, you cannot use generic profiles to specify profiles to the PTKTDATA class. If you specify a generic profile, it is ignored during PassTicket processing for the application, and PassTickets cannot be used to authenticate users for that application.

7.7.5 Protecting the RACF Secured Signon Application Keys

When you define the RACF Secured Signon application keys, RACF either masks or encrypts each key. If the system has a cryptographic product installed and available, you can encrypt the application keys for added protection.

Note: Be sure the universal access authority (UACC) of the RACF database is *NONE*. This prevents unauthorized users from listing or copying the RACF data set that contains the sensitive RACF Secured Signon application keys.

7.7.5.1 Masking the RACF Secured Signon Application Keys

If the system using the RACF Secured Signon function does not use a cryptographic product, RACF masks the key with a proprietary masking algorithm when you define or alter it. The masking algorithm that masks the application keys while they reside on the RACF data base is an IBM proprietary algorithm. It resides within in the RACF object code portion of the RACF program product and is designed to provide protection against casual viewing of the RACF Secured Signon masked keys. The algorithm is not a cryptographic algorithm and cannot provide the level of security for the RACF Secured Signon keys that use of cryptography can provide.

To mask the application key when you define or alter it, use the SSIGNON operand and KEYMASKED suboperand with the RDEFINE or RALTER command.

7.7.5.2 Encrypting the RACF Secured Signon Application Keys

Using a cryptographic product to encrypt the RACF Secured Signon application keys ensures maximum possible security for the application keys.

With a cryptographic product, such as the ICRF/ICSF, RACF can store the keys on the database in a encrypted form. RACF uses the functions of the cryptographic product to be sure the encrypted keys do not exist in clear-text form within the system main storage for RACF processing, except when they are being defined. Therefore, if a system storage dump occurs, the keys are not exposed in the dump.

To encrypt the application key when you define or alter it, use the SSIGNON operand and KEYENCRYPTED suboperand with the RDEFINE or RALTER command.

7.7.6 PassTicket Validation Process

There is no way for RACF to determine whether a PassTicket is sent to RACF for validation instead of a password. The RACINIT pre- and post-processing exit still gets control, but also there is no way to see the difference between a PassTicket and a password.

RACF validates a password or PassTicket with the following steps:

1. Determine whether the value in the password field is a valid RACF password. If it is a valid password, the validation is completed. If the value is not a valid RACF password for this user ID, processing continues.
2. Determine whether a RACF Secured Signon application profile has been defined for the application in the PTKTDATA class. If a profile has not been defined, RACF sends a message to the user ID indicating that the password is invalid. If the application is defined to the PTKTDATA class, processing continues.

3. Evaluate the value entered in the password field. The evaluation determines whether:
   - The value is a PassTicket consistent with this user ID, application, and time range
   - If it has been used previously on this computer system for this user ID, application, and time range

   Note: A PassTicket is considered to be within the valid time range when the time of generation, with respect to the clock on the generating computer, is within 10 minutes (plus or minus) of the time of evaluation with respect to the clock on the evaluating computer.

   If the value is determined to be a valid PassTicket, the user is allowed to access the desired application. If the value is not a valid PassTicket, RACF sends a message indicating that the user entered an invalid password.
4. Give the user ID access to the desired application if the PassTicket is valid.

7.7.7 Using the Callable Service to Generate a PassTicket

In a host environment, the authentication server, for example NV/AS, can call the RACF Secured Signon callable service to build a PassTicket.

The RACF Secured Signon callable service:

- Is branch-entered by callers.
- Is *not* supported in cross-memory mode. Access register (AR) mode must use address space control (ASC).
- Uses standard linkage.
- Uses the current system time (GMT) as input for the algorithm.
- Returns the PassTicket in general purpose register 0 (the leftmost four characters) and general purpose register 1 (the rightmost four characters).
- Provides the following return codes:
  - Return code 0 in register 15 if the PassTicket is produced.
  - Return code 8 in register 15 if the PassTicket is not produced.

Before calling the RACF Secured Signon callable service, the application must locate the address of the service. You can find this address from field RCVTPTGN in the RACF communication vector table (RCVT). The ICHPRCVT macro maps the RCVT, and field CVTRAC points to it in the MVS communications vector table (CVT).

To be sure the label RCVTPTGN resolves, APAR OY65283 must be installed on the MVS system that is used to compile the program to generate PassTickets.

No additional recovery processing is provided by the RACF Secured Signon callable service beyond what is already in effect within the invoking program.

This program with the callable service must be executed from an authorized program facility (APF) library.

You can use the following example of a generalized programming technique with System/390* Assembler language to invoke a callable service:

```
PTKTGEN  DS    0H
         MODESET KEY=ZERO,MODE=SUP
         L     R15,16                  Get pointer to CVT
         USING CVT,R15                 Make CVT addressable
         L     R15,CVTRAC              Get pointer to RACF CVT (RCVT)
         USING RCVT,R15                Make RCVT addressable
         L     R15,RCVTPTGN            Get pointer to PassTicket rtn
         CALL  (15),(USERID,APPNAME)   Call PassTicket generator
         LTR   R15,R15                 Test return code
         BNZ   RETURN                  No, exit
         STM   R0,R1,PTKTFLD           Store PassTicket in PTKTFLD
         MODESET KEY=NZERO,MODE=PROB
```

Where:

USERID Is the RACF user ID of the user the PassTicket authenticates. This field is a maximum of nine bytes. The first byte contains the length of the nonblank portion of the USERID field that follows. Bytes 2 through 9 contain the user ID and must be in uppercase and left-aligned in the field.

APPNAME Is the name of the PTKTDATA profile that the RACF Secured Signon function uses to locate the secret application key used in the PassTicket generator algorithm. (Refer to Sections 7.7.3, “Defining PTKTDATA Class Profiles” and 7.7.4, “Determining Profile Names in the PTKTDATA Class” on page 69). This field is a maximum of nine bytes. The first byte is the length of the nonblank portion of the APPNAME field that follows. Bytes 2 through 9 contains the application name and must be in uppercase and left-aligned.

To generate a PassTicket without using the RACF callable service, you must incorporate the RACF PassTicket generator algorithm into your program. The RACF PassTicket algorithm consists of two parts:

- The RACF PassTicket generator.
- The RACF PassTicket time-coder. The time-coder is invoked from within the RACF PassTicket generator and returns its results to the generator.

Enhanced Exploitation of RACF 1.9.2 - Secured Single Signon Using NV/AS provides a sample program to test the PassTicket callable service. This program does the following functions:

1. Locates the address in the Link Pack Area (LPA) of the RACF Secured Signon PassTicket generate routine.
2. Sets up a parameter list for the PassTicket generator that contains:
   - The RACF user ID of the user that the PassTicket is being requested for (not necessarily the current user)
   - The name of the application that the PassTicket will be used for
3. Branches to the RACF PassTicket generator routine. If the parameter list information is correct, the PassTicket is returned to this routine in registers 0 and 1.

Chapter 8. Solution Security Features

The basic idea in using a Personal Security card to hold the key and to generate the PassTicket is that since the workstation is fundamentally untrusted, another secure device is needed. The user then has a formal interface to this device, and can use only that.

The Personal Security card employs a single electronic chip that contains an 8-bit computer and memory. Although it is theoretically possible to probe the chip to obtain information stored within the memory, such as the secret application key, an attacker would need confidential design information from several companies and very advanced semiconductor laboratory test equipment. The physical design of the Personal Security card has been rated highly tamper resistant.

The logical design of the Personal Security card also provides protection for the stored data and its processing capabilities. The Personal Security card′s comprehensive and flexible access control system provides a secure processing environment. It is essential that application designers understand the access control system and employ it appropriately so as to secure access to sensitive data, keys, and processes.

Note: The sample profiles and configuration data that are shipped with the standard products do not provide sufficient security. They should never be used in real applications.

This solution tries to go as far as possible and to obtain as much security as possible without going to expensive tamper resistant modules. The authors see no other exposure apart from the lack of tamper resistancy in protection of the secret keys. Specifically, even the factory doing the microcode initialization is not able to reveal the secret data on the Personal Security card.

A class of attacks concentrates on the case where one user finds or steals another user′s card but does not know the PIN codes. Here it is important to know that when no user is signed on to a Personal Security card, the first profile (usually PSCUSER0) is valid (as public profile), but commands with the Initial Verification Required bit are unavailable. This bit is not set for any command in the sample files distributed with TSS. Another class of attacks concentrate on what the valid user PSCUSER0 may do; could he somehow generate a PassTicket for another user ID exploiting the existing keys on the card?

8.1 What If I Lose My Card?

Suppose somebody finds a valid card belonging to another user. Is it possible to logon to the real owner′s user IDs by having the card generate PassTickets for them?

Of course the normal signon application requires a PIN verification, which the finder is supposed to be unable to perform. But the finder could write a workstation program like the signon application and just skip the part where a PIN verification is done. But now the attacker will be rejected by the ″Initial Verification Required″ bit in the Command Configuration table on the card since no user is verified for this card.

8.2 Can I Change Data Blocks on the Card?

It may be possible for a user to execute the *Card_Block_Clear* (CSUASCC) verb to de-allocate all data blocks on the card. This does not remove the secret application keys in the registers. The user might then add his own Application Data Block (APLBLKii) which would effectively replace one the administrator had created (and which would point to an existing secret application key register ii).

The user could also add a block, APLBLKxx, and let it point to register ii, where xx differs from ii. This would not conform to the block naming convention, but that is not required. However, the block token should be the MAC of the block contents, using the secret key, and it is impossible for the user to generate such a MAC since the secret key on the card can only be used for a MACVER. The control vector prohibits MAC generate and export of the key. Moreover these services are even excluded from the card's microcode.

The *PassTicket_Generate* (GCSBNPTG) verb executing on the card will enforce such a MAC verify returning return code 8 and reason code 95 rather than a PassTicket, should the verify fail.

8.3 Can I Retrieve the Keys From a Card?

There is no way to get the keys out of the Personal Security card register and into, for example, application storage on the Personal Security card or a key register inside a 4754 or a 4755.

This is so, even at the lower level device interface from the device driver. So it will not help to design your own smart card reader, and access the card through it. Such a card reader will not provide any capabilities over the existing readers.

8.4 Can I Tap the Algorithm Inputs?

Note that the user ID and application ID are not parameters of the CSNBPTG verb. Instead, you pass the name of an existing APLBLKii block and the verb finds the user ID and the applid inside the block. The verb bypasses normal requirements for read authority and token knowledge when accessing the block, but it enforces the MAC check instead. Thus, one cannot feed the algorithm with any user ID unless one can generate MACs using the secret key (and that is prohibited by the control vector).

8.5 Can I Modify the PassTicket Algorithm?

The PassTicket algorithm and its input data (user ID and application ID, but not time) resides in a part of the EEPROM which is logically separate from the data blocks, or any other user-accessible structures on the card, and cannot be modified.

The input data is not in memory of the workstation or client machine, and cannot be altered there either. The algorithm is also not in workstation or client machine memory and is not alterable.

It is theoretically possible to load new microcode (supposing that the command IML Enable is available in the Command Configuration), but this will destroy the existing configuration, data blocks, and key registers.

8.6 Can I Calculate the Secret Key?

A user can load a new key into a register. Invoking the CSNBPTG verb for an application that uses this Personal Security card register, he will be told via the MAC check (return code 8 and reason code 95) whether the MAC was OK or not. The user can continue this until he has a MAC verification. This will require on average 2**55 keyloads+CSNBPTG operations, which is considered prohibitive.

However, note that no auditing of such an exhaustive attack will take place.

8.7 What if the Administrator Makes Errors?

An administrator may have made mistakes, like removing the PassTicket generate ″Initial Verification Required″ bit or allowing removal of the bit without verification. This way a clever user may load a new Command Configuration table without verification requirements, and then a found card could be misused.

An administrator who has other versions of the Personal Security card (for example, a standard card) may have written the data blocks to a wrong kind of Personal Security card. This card will not be able to run CSNBPTG (because the microcode of other cards do not include this verb) so it does not matter whether the command (x′33′) has verification required or not. And the control vectors still prohibit use of the keys although more services are available, like Key_Export.

8.8 What If I Forget My Card?

Valid owners of Personal Security cards will sometimes forget their cards. Then they cannot logon to their normal user IDs with PassTickets. In this situation, the best solution is to logon with user ID and password and have the security administrator set new passwords both before and after the logon.

If the passwords are reset after the user has logged on and the person doing the reset is located centrally, a wiretap on the forgetful user will reveal his new clear password, but an additional reset of the password makes it difficult to misuse this knowledge. Remember that a RACF user will have to change his password if the administrator has reset it.

Chapter 9. What is the Time?

The PassTicket generator and validation algorithm will get its “time” input from the local host, PC, or RS/6000 clock. It is likely that the computer that validates the PassTicket is not the same computer that generates it. To provide for differences in their internal clocks and for network delays, the algorithm allows the generated time to be 10 minutes on either side of the CPU clock of the computer that is validating the PassTicket.

It is also possible that the computers are not in the same time zone. That does not prevent the usage of PassTickets since the time stamp is calculated from GMT and not from the local time. This chapter describes how to check the GMT in the participating environments.

9.1 Host Time

To check the GMT time in the participating MVS environments, enter the MVS command **Display T** on the various MVS systems. The output of this command looks like:

```
IEE136I LOCAL: TIME=15.57.42  DATE=1993.300  GMT=19.57.42
DATE=1993.300
```

The local time is calculated from the hardware CPU clock, which uses GMT, and an offset that is specified in the CLOCKxx member of SYS1.PARMLIB by the TIMEZONE keyword. The **Set CLOCK=HH.MM.SS** command allows you to set the local time. However, this does not affect the GMT time that is used to calculate the PassTicket.

The GMT can be changed only at IPL time. If the OPERATOR(PROMPT) parameter is included in the active CLOCKxx member of SYS1.PARMLIB, the system automatically issues the message IEA888A once the system has been initialized. The system displays the time and date and gives you the option of accepting or changing them as follows:

```
00 IEA888A GMT DATE=1993.300.CLOCK=19.22.31
   IEA888A LOCAL DATE=1993.300,CLOCK=15.22.31 REPLY U, OR GMT/LOCAL TIME
```

If the values are acceptable, reply “U.” If you want to change the value of the Time Of Day (TOD) clock, enter a new date, time, or both as follows:

```
R 00,CLOCK=21.22.00,GMT
```

If you specify a different TOD clock setting, message IEA903A is issued asking you to press the TOD clock security switch. Reply “U” to the message IEA903A and, at the exact time that matches the TOD clock setting, depress the TOD clock security switch. Once you have successfully set the TOD clock, the system displays the time and date and gives you the option of accepting or changing them by issuing the message IEA888A again.

For more information about synchronizing the clocks in, for example, a sysplex environment, refer to *MVS/ESA Initialization and Tuning Reference* and to *MVS/ESA System Commands*.

9.2 Workstation Time

The PC program uses the C language library function *gmtime* to obtain the time-of-day. This routine is intended to return the Universal Time (sometimes called "Greenwich Mean Time"). The *gmtime* library function looks for an environmental variable named *TZ*, which should specify the offset of the local time from GMT. (You set your PC time to your local time, of course.) If the environmental variable is not found, the C function assumes your PC time is set for Eastern Standard Time, which is five hours earlier than GMT. Thus, it will add five hours (making adjustments for the date) to your local PC clock and return this time.

The algorithm used to compute a password assumes the application server is using the same time of day. Both routines (the client machine or workstation and the server) request GMT from their operating systems. You may need to enter an environmental variable for your DOS system. This is done with a command like:

```
SET TZ=PST8
```

which would tell *gmtime* that your PC's clock is set for Pacific Standard Time, eight hours earlier than GMT. The *gmtime* function would then make the appropriate adjustment before returning the time to the HST program.

The interaction between *gmtime* and the TZ variable is not too well documented. It appears that the three characters are ignored. The digit that follows the three characters is the only effective character. If it is positive, the PC's time is earlier than GMT (that is, "west" of GMT). If the time is negative, the PC's time is later than ("east" of) GMT. Thus times like the following are valid:

```
SET TZ=XXX5        (makes U.S. Eastern Standard Time)
SET TZ=EST5        (makes U.S. Eastern Standard Time)
SET TZ=CET-1       (makes Central European time, +1 hour from GMT)
```

The server must be set up such that a TIME macro will return the correct GMT. Of course, the PC program and the server do not really require GMT. However, they do require the *same* time. Requesting and using GMT is one way to accomplish this. Once you determine the correct operand to match your personal computer's GMT to your server's GMT, you should put the appropriate SET command in AUTOEXEC.BAT.

Appendix A. IBM Cryptographic Facilities Highlights

This appendix provides an overview of the currently available IBM cryptographic facilities.

Components of previous security systems were designed independently from one another and were often difficult to integrate. The IBM Transaction Security System and the Integrated Cryptographic Facility implements the IBM Common Cryptographic Architecture (CCA) and offers a comprehensive set of security products that allow users to implement end-to-end secure systems with IBM components.

A.1 Cryptographic Application Program Interface

The are several cryptographic Application Program Interfaces (APIs) defined. This section provides a list of the APIs that are implemented in the cryptographic facilities described in this appendix.

A.1.1 Common Cryptographic Architecture

The IBM Common Cryptographic Architecture: Cryptographic Application Program Interface (API) is a set of specifications that establishes a unified data security architecture applicable to the design and implementation of a wide range of security products.

The cryptographic API describes a set of cryptographic services that provide:

- Data privacy
- Data integrity
- Cryptographic-key installation and generation
- Electronic cryptographic-key distribution
- Personal Identification Number (PIN) processing

The architecture provides the cryptographic separation required for good security.

Some of the IBM Common Cryptographic Architecture compliant products have implemented functions beyond the scope of the Common Cryptographic Architecture. Refer also to Section A.1.3, “Security API” on page 81. For example, facilities can provide additional specification options for some parameters and, sometimes, parameters besides to those defined in the cryptographic API services. The PassTicket generate call that is used in the Secured Single Signon, as described in this document, is a good example of additional functionality.

Generally, these specifications are provided through additional callable services. However, a program using these services does not adhere to the IBM Common Cryptographic Architecture API, and may have to be modified before it can run with other cryptographic products that follows the Common Cryptographic Architecture API conventions.

Note: The PassTicket call is only available on a specific version of the IBM Personal Security card. The PassTicket generate call does not adhere to the IBM Common Cryptographic Architecture API.

The cryptographic services may be used by either a customer application or a system service to achieve security objectives. The cryptographic API provides sufficient detail to allow an application programmer to use IBM Common Cryptographic Architecture compliant products. It supports portability for applications across IBM systems that have access to IBM Common Cryptographic Architecture compliant products.

The cryptographic API provides languages, commands, and calls that programmers can use to develop applications that take advantage of the consistency offered by the IBM CCA. The cryptographic API callable services are listed in Table 1.

Table 1. Cryptographic API Services

Data Operations	*Key Management*	*PIN Management*
Encode Decode Encipher Decipher Ciphertext Translate MDC Generate MAC Generate MAC Verify	Clear Key Import DATA Key Export Key Export Key Generate Key Import Random Number Generate Secure Key Import	Clear PIN Generate Encrypted PIN Translate Encrypted PIN Verify

An example of a callable service format can be:

```
CALL CSNBENC(
             return_code
            ,reason_code
            ,exit_data_length
            ,exit_data
            ,key_identifier='internal_key_token'
            ,text_length=8
            ,plain_text='application_key'
            ,initialization_vector=ICV
            ,rule_array_count=1
            ,rule_array=CBC
            ,cipher_text='encrypted_appl_key
```

The cryptographic API is not in itself a product or a piece of code; but, as a definition, it establishes a common base across multiple environments.

A.1.2 CCA Public Key Algorithm Tower

The IBM Common Cryptographic Architecture: Public Key Algorithm (PKA) Tower Cryptographic Application Program Interface (API) is a set of specifications that defines four optional extensions to the definition of the of the IBM Common Cryptographic Architecture: Cryptographic API. The PKA Tower further delineates a unified data security architecture applicable to the design and implementation of a wide range of security products. The PKA Tower describes a set of cryptographic services that provide support for digital signatures and distribution of Data Encryption Algorithm (DEA) key encrypting keys.

The PKA Tower cryptographic API is not in itself a product or a piece of code. But as a definition, it establishes a common architecture definition across multiple platforms and environments. The PKA Tower is not intended to be a stand-alone architecture. If an implementation implements one of the extensions

of the PKA Tower, then it must also implement the base Common Cryptographic Architecture Cryptographic API.

Currently, the IBM Transaction Security System implements PKA Tower extensions described in *IBM CCA: Cryptographic Application Program Interface Reference - Public Key Algorithm*.

A.1.3 Security API

The security application programming interface (SAPI) is the primary application programming interface for accessing the services provided by the Transaction Security System hardware products in a workstation and an MVS host.

The security API extends the IBM Common Cryptographic Architecture (CCA) by providing additional services and additions to the existing CCA services.

The security API is designed for high-level languages, such as COBOL, PL/1, Pascal, or C, and for low-level languages, such as Assembler. It is also designed to enable the user to use the same verb entry point names and variables in the DOS, OS/2, AIX for RISC System/6000, and MVS environments. (The term "*verb*" implies an action that an application program can initiate; other systems and publications might use the term *callable service* instead of *verb*.)

A.2 Cryptographic Algorithms

A cryptographic algorithm is a set of rules that specify the mathematical steps needed to encipher and decipher data. There are many cryptographic algorithms defined. This section lists the cryptographic algorithms that are implemented in the cryptographic facilities described in this appendix.

A.2.1 Data Encryption Algorithm

For commercial business applications, the cryptographic process known as the Data Encryption Algorithm (DEA) has been widely adopted. The *Data Encryption Standard*, and other documents, define how to use the DEA to encipher data. Many other processes for concealing data, such as protection of passwords and Personal Identification Numbers (PINs), are based on the DES process. (The DEA is often referred to as the *DES algorithm* or just as the *DES*.)

The DES algorithm uses a key to vary the way that the algorithm processes the data. A DES key is a very small piece of data (56 bits) that is normally retained in eight bytes. The same key is used to transform the original data (plaintext) to its disguised, enciphered form (ciphertext) and to return it to its plaintext form. Because the DES algorithm is common knowledge, you must keep the key secret to make the data confidential; otherwise, someone who has the key that you used to encipher the data would be able to decipher the data. *Key Management* refers to the procedures that are used to keep keys secret.

Because the same key is used to both encipher and decipher the data, the process is said to be *symmetric*, and it uses a symmetric key.

A.2.2 Commercial Data Masking Facility

The Commercial Data Masking Facility (CDMF) is a scrambling technique for data confidentiality and is intended as a substitute for the Data Encryption Algorithm in cryptographic products exported by IBM to customers who are unable to receive DEA-based products because of U.S. government export regulations. Although subject to USA Department of State jurisdiction, CDMF-based cryptographic facilities may be freely exported to any customer in most of the countries of the world.

A CDMF key is 64 bits, of which 56 determine the data scrambling transformation and eight which may be used for key parity. However, a CDMF key has an effective strength of 40 DEA-key bits.

A.2.3 Public-key Systems

In another type of cryptographic process, an *asymmetric* process, one key is used to encipher the data, while a different, but corresponding key, is used to decipher the data. A system that uses this type of process is known as a *public-key* system. The key that is used to encipher the data is widely known, but the corresponding key for deciphering the data is secret. For example, many people who know your public key can send enciphered data to you confidentially, knowing that only you should possess the secret key for deciphering the data. Public-key algorithms have been incorporated into processes for simplifying the distribution of secret keys and for assuring data integrity.

Unfortunately, the widely known and tested public-key algorithms use a relatively large key and use even more computer time than the DES algorithm. The use of a public-key system is, therefore, often restricted to situations in which the characteristics of the public-key algorithm have special value.

One widely known public-key algorithm, the Rivest, Shamir, and Adleman (RSA) algorithm, has, like the DES algorithm, been the subject of considerable testing and research. The results have been reported in the public domain; the RSA algorithm is, therefore, understood and accepted.

A.3 Hardware Components

The IBM Common Cryptographic Architecture compliant products include the following hardware components and the complementary supporting software:

- IBM Integrated Cryptographic Facility (ICRF) on selected IBM mainframes
- AS/400 Cryptographic Processor
- Host channel-attached IBM 4753 Network Security Processor
- High-performance IBM 4755 Cryptographic Adapters for:
  - Personal computer, both Micro Channel* and Industry Standard Architecture (ISA)
  - RISC/6000 POWERstation* and RISC/6000 POWERserver*
- RS-232 attached IBM 4754 Security Interface Unit for:
  - Personal computer
  - AS/400
- Credit-card-size, state-of-the-art, Personal Security card

The application programming interface provides common programming in the host and the workstation, and supports most of the Systems Application Architecture (SAA*) languages (except REXX and RPG).

The older IBM cryptographic products, 3848/CUSP and PCF, do not adhere to all applicable standards of the American National Standards Institute and the International Organization for Standardization because such standards were nonexistent when the products were developed. The IBM Common Cryptographic Architecture compliant products were specifically designed to meet applicable ANSI and ISO standards. They also provide a common base for the future development of related products and applications.

Note: Refer to *IBM CCA: Cryptographic Application Program Interface Reference* and to *IBM CCA: Cryptographic Application Program Interface Reference - Public Key algorithm* for a list of supported standards.

IBM announced enhanced MVS/ESA security by providing cryptographic functions for data secrecy, data integrity, personal identification, and key management. These functions are provided in the IBM Common Cryptographic Architecture compliant products through the combination of secure hardware, and the supporting software. PCF is a ″software-only″ solution, which means that the DES algorithm and the cryptographic keys are available in the storage of the processor on which PCF is running.

Cryptographic services are tailored for the environment in which they will operate. However, they should perform the same operation, with the same result, regardless of the cryptographic facility that is used or the environment. If a customer uses a workstation-based product for end-user cryptographic services, it should be compatible with host-based products that provide the same services. In addition, if a customer writes an application that requires cryptographic services, the application should be portable between the IBM strategic operating systems.

The above considerations led to the development of the Common Cryptographic Architecture (CCA) - Application Program Interface (Cryptographic API).

It should be clearly understood that only certain cryptographic facilities, such as ICRF and Integrated Cryptographic Service Facility/MVS, and IBM Transaction Security System, support the CCA. The new products also support the services that were available in the earlier host products 3848/CUSP and PCF.

A.3.1 IBM TSS Workstation Products

The IBM Transaction Security System products provide comprehensive support for Data Encryption Standard (DES)-based, CDMF-based, and public-key-based cryptographic processing. The hardware products feature tamper-resistant mechanical and electrical designs that are combined with a sophisticated set of access controls. Together, these products create a secure subsystem.

The Transaction Security System products are supported in major computing environments where applications and system programs access the hardware services through a programming interface that is common across the environment. The software support consists of access methods and utility programs that help the user to set up the system and perform basic cryptographic key-management functions.

Figure 20 shows an overview of the Transaction Security System products.

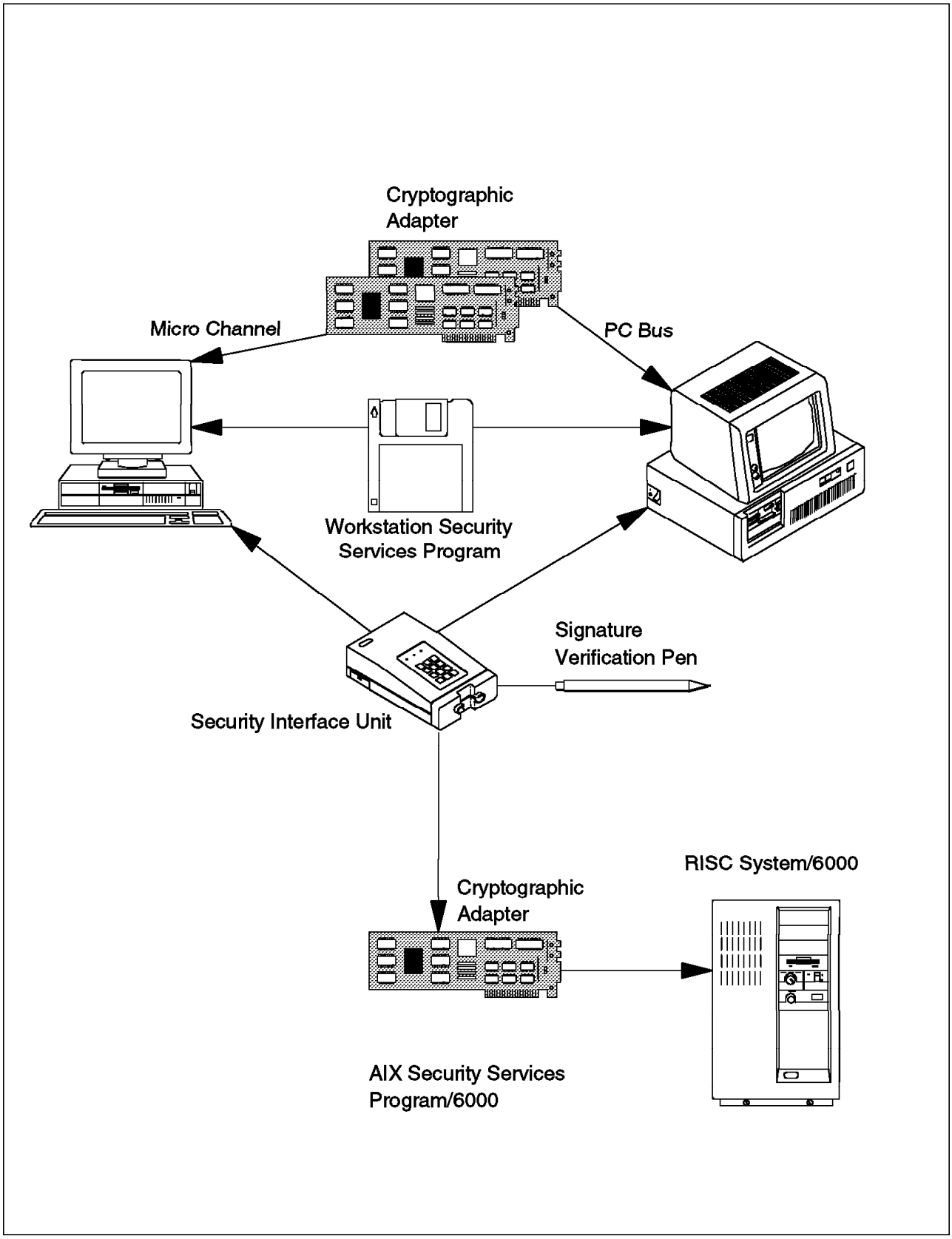

Figure 20. Transaction Security System Workstation Products

A.3.2 IBM 4755 Cryptographic Adapter

Several models of the cryptographic adapter are available that support a broad range of DES, CDMF, and public-key cryptographic processes. These cryptographic processes are performed within a highly secure module that is mounted on the adapter. The adapter can be used in DOS, OS/2, and AIX environments.

A.3.3 IBM Personal Security Card

The IBM Personal Security card (PSC) is a single-chip, integrated-circuit card with an 8-bit processor and nonvolatile storage. The card looks like a typical credit card; however, it has many additional features, such as the following:

- DES-based cryptographic processing
- Access controls that authorize unique functions for up to four users
- Over 4000 bytes of portable, access-controlled, nonvolatile data storage

Refer to Chapter 4, “IBM Personal Security Card” on page 19 for a comprehensive overview of the Personal Security card.

A.3.4 IBM 4754 Security Interface Unit

The security interface unit can be used in the DOS, OS/2, and AIX environments. The unit is a cryptographic facility that can be connected to a cryptographic adapter in a personal computer or in a RISC System/6000 workstation. The unit can also be connected to an RS-232C serial port in a personal computer.

The security interface unit is designed primarily to permit communication between an application program and a Personal Security card. When a cryptographic adapter is not required in a configuration for performance or functional reasons, the unit can also serve as the principal cryptographic facility in the system.

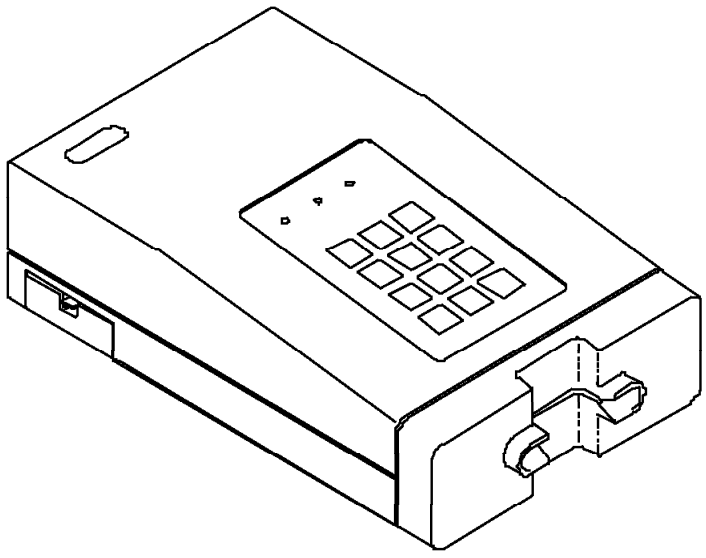

Figure 21. IBM 4754 Security Interface Unit

The security interface unit contains the following components:

- A secure, tamper-resistant DES-cryptographic facility that can provide primary cryptographic support when a cryptographic adapter is not available in the personal computer.
- An input/output port (or reader) for communicating with a Personal Security card.
- A 12-key keypad that is used to enter the Personal Identification Number (PIN). The PIN is encrypted and sent to the Personal Security card for verification.

  The keypad can also be used by the application program. There's an API call available to get keystroke data from the SIU for use in the calling application.
- A battery-powered, secured key-storage area and a clock calendar. Specific authority is required to set (and read) the clock calendar. This secure clock can be used to limit the periods when the access-control profiles on the Personal Security card and the cryptographic adapter can be activated.
- The interface for the (optional) Signature Verification feature.

A.3.5 Argus/200 Smart Card Acceptance Device

The test described in this document is performed with a Argus/200 smart card acceptance device. This device allows the use of the Personal Security card at workstations and client machines. The device is connected via an RS-232 serial port.

Device drivers are used to interface the Argus/200 smart card acceptance device and the Personal Security card to the operating system and the Workstation Security Service Program security server and application programs, such as the signon application.

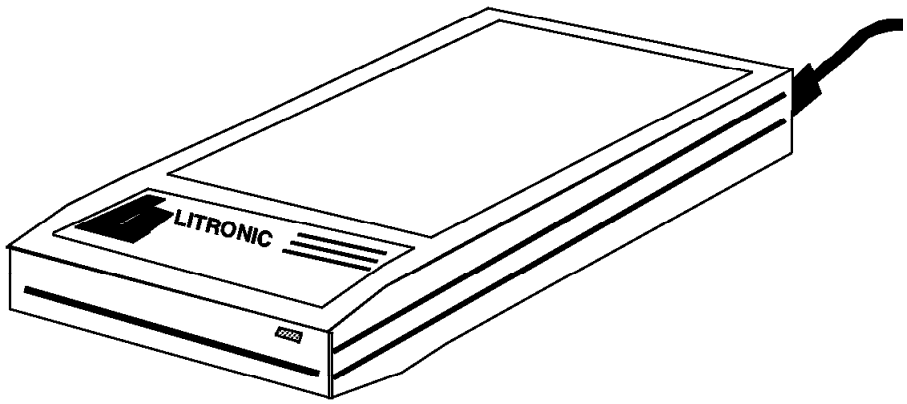

Figure 22. Argus/200 Smart Card Acceptance Device

A.3.6 Signature Verification Feature

The Signature Verification feature can be used to authenticate users. This feature includes the signature verification pen, which is attached to the security interface unit by a flexible cable, and the signature verification module, which is mounted on the cryptographic adapter. The Signature Verification feature determines the authenticity of the user's signature by measuring the acceleration and the pressure that the user applies to the pen when writing a signature. These pen movements and pressure are digitized, sampled, encrypted, and sent to the signature verification module. The module determines the validity of the signature by comparing its characteristics to the characteristics of previously entered signatures that are stored in data blocks on the Personal Security card.

A Personal Security card can store four profiles, each of which contains data for one user. Only one profile for each card can store signatures.

A.3.7 Workstation Software

There are two workstation software programs available to support cryptographic hardware:

- The IBM Workstation Security Service Program is used when a cryptographic adapter or a security interface unit is installed in a personal computer with DOS or OS/2.
- The IBM AIX Security Services Program/6000 is used with the cryptographic adapter in a RISC System/6000 system.

These workstation software products provide functions such as:

- Security API Libraries. The software provides *include files* that can be used in application programs for calling the various cryptographic functions.
- Hardware-initialization and key-management utilities. These utilities provide many different services that use the cryptographic adapter, the security interface unit, and the Personal Security card.
- Device drivers for the security interface unit and the cryptographic adapter.
- Sample programs and access control data.

A.3.8 Access Control

Each Transaction Security System hardware component maintains a logical access-control regime. All functions that each component can perform are divided into a set of individual *commands*. When an individual command is performed in the same manner on more than one component, the same command is used with each device.

A hardware component is initialized with a definition of which commands it can perform. A set of criteria is established for each command to define the circumstances under which it can be performed. This information is contained in the *command configuration data* register.

Each defined user has a *profile*. A "public" profile also exists for when no user has been authenticated. The user's profile is also initialized in the hardware; this profile defines how a user is to be authenticated (PIN, password, or signature), when a user can be authenticated (time of day and day of week), and when a user's authorization expires. The profile includes the user's

authorization level and whether the user is authorized to perform each command.

When a user is authenticated to the hardware, the hardware operates under the combined restrictions of the user′s profile and its own configuration.

A.4 Authorizing Hardware Commands

The security server separates a verb request into commands and sends each command to a cryptographic facility, for example the Personal Security card. Before processing a command, the hardware component, in this case the smart card, does a series of access-control tests to ensure that the command is authorized for that hardware component.

The information that the hardware uses in the access-control tests is contained in the following access-control registers:

- Command configuration data register
- Profile registers
- Active profile indicator
- Pre-execution verification OK indicator
- Holiday table register
- Date and time register

The workstation security service utility is used to create the information for all registers. Various utilities are provided to load the information into the registers.

Chapter 4, “IBM Personal Security Card” on page 19 provides an overview of access-control facilities in the TSS environment.

A.5 Hardware Initialization and Key Management Utility

Two Hardware Initialization and Key Management (HIKM) utilities are provided:

- The DES Hardware Initialization and Key Management utility
- The public-key Hardware Initialization and Key Management utility

You can use the DES HIKM utility to do the following:

- Create and edit user profiles and device configuration information
- Manage DES cryptographic keys
- Create, read, and write Personal Security card data blocks

You must use this utility to create configuration information for the Transaction Security System hardware.

A.6 Host Cryptographic Facilities

Host cryptographic facilities consist of the following hardware and software products:

- IBM 4753 Network Security Processor
- IBM 4753 Network Security Processor MVS Support Program
- Integrated Cryptographic Facility
- Integrated Cryptographic Service Facility/MVS

When a cryptographic facility is available on the host, RACF can store the secret application keys, that are needed to generate PassTickets, encrypted under the master key of this cryptographic facility.

The following sections give high-level overviews of these host cryptographic facilities.

A.6.1 IBM 4753 NSP and NSP MVS Support Program

This channel-connected cryptographic-processing I/O unit uses the cryptographic adapter to provide a comprehensive set of DES-based, public-key-based, and CDMF-based cryptographic services. Multiple processors can be connected to a System/390 processor or to a System/370* processor that uses the MVS operating system. The network security processor connects to a host through a block-multiplexer channel.

The IBM 4753 Network Security Processor MVS support program provides an access method for the network security processor to use with MVS operating systems. The access-method program modules reside in an authorized program facility (APF) library in MVS. The MVS support program allows host application programs to request DES-based, CDMF-based, or public-key (RSA)-based cryptographic services in the network security processor.

The MVS support program runs in a System/390 processor or in a System/370 processor under various MVS operating systems. It can support one or more network security processors.

A.6.2 ICRF and Integrated Cryptographic Service Facility/MVS

The Integrated Cryptographic Facility (ICRF) is an extension to the base of selected IBM mainframe processors. The ICRF provides an efficient implementation of the DES that delivers essential cryptographic services and key management functions. The overall design objective is to provide a DEA-based cryptographic facility on System/390 and ES/9000* mainframes to support bulk encryption and financial transaction environments with high performance and security and various levels of compatibility.

The ICRF design is based on the basic security, integrity, auditabilty, and scalability of MVS/ESA running in an IBM mainframe.

The ICRF supports the IBM Common Cryptographic Architecture.

The Integrated Cryptographic Service Facility/MVS (ICSF/MVS) is the support software for the Integrated Cryptographic Facility. ICSF/MVS enhances the MVS/ESA security by providing cryptographic functions for data security, data integrity, personal identification, and key management.

A.7 Security Functions

To help you to select the products and services that you need to implement a data security policy, IBM has categorized the following security functions according to standards of the International Organization for Standardization These security functions are:

- Identification and authentication

  Identifies users to the system and provides proof that they are who they claim to be.

  A Personal Identification Number (PIN) is an authorization mechanism to prove that individuals are really who they claim to be. A PIN is similar to the concept of a password except that a PIN, as known by an individual, is normally composed of decimal digits and is typically from four to six digits in length. Today, most PIN processing is done in connection with an Automatic Teller Machine (ATM) and authorizes personal financial transactions.

  A customer of a financial institution inserts their ATM card into an ATM to identify who they are and then enters a PIN to prove their authorization to perform financial transactions. The ATM transmits the information from the magnetic strip on the ATM card and the supplied trial PIN to a site that is authorized to verify the trial PIN.

  PIN generation schemes commonly use the DEA and a secret key. There are two basic methods of verifying a PIN, the PIN database method and the PIN calculation method, both of which use the DEA.

- Data confidentiality

  Protects sensitive data from unauthorized disclosure.

  If confidential data, for example an application key, is moved between nodes through a network, the only possible protection is by encryption. VTAM Session Level Encryption (SLE) offers the capability to establish a session between two end points (LUs), where the user data is encrypted by the sending LU and decrypted by the receiving LU.

  The role of sender and receiver may change during the session. User data is encrypted under the session key, which is dynamically generated at one end of the session, and used only for the lifetime of the session.

  The encrypted session key is transferred at session setup time from the initiating LU to the receiving LU. The session key is encrypted under a key-encrypting key that must be known by both parties in the session.

  SLE operates end-to-end, such that until the data is decrypted by the receiving (or secondary) LU; it will never appear in the clear on either intermediate systems or links. Only the user data is encrypted; the control data remains in the clear.

- Data integrity

  Ensures that data is in its original form and that it has not been altered.

  When you want someone to be able to confirm the integrity of your data, you can use, for example, the DES algorithm to compute a Message Authentication Code (MAC).

This is a very powerful tool; it is almost impossible to meaningfully modify the data and still have it produce the same MAC for a given key. The standardized approaches authenticate data such as financial transactions.

After a MAC has been computed, it is sent with the data. To authenticate the data, the system uses the DES algorithm to recompute the MAC; the system then compares this result with the MAC that was sent with the data.

- Nonrepudiation

  Assures that the message was sent by the appropriate individual.

  A useful feature of public-key cryptography is the ability to generate and verify digital signatures. From a conceptual viewpoint, a digital signature serves essentially the same purpose as a handwritten signature.

  A digital signature is applied to a message when it is sent and is verified by the receiver of the message. In this way, the receiver is virtually certain that the message originated with the presumed sender.

  The sender can create a digital signature by deciphering a message, or (more usually) a function of a message, with his private key. Successful encryption of the signature by the receiver, using the sender′s public key, is reasonable verification of the sender′s identity, since it is assumed that only the presumed sender has access to the private key that matches his own public key.

The IBM cryptographic facilities, both for workstations and for the host, can be used to implement these security functions in installations security policy.

A.8 Export Regulations

Many governments have laws that relate to exporting cryptographic implementations across national borders. These laws relate to both software and hardware implementations of DES-based and public-key-based cryptographic processing and other cryptographic implementations.

The IBM products described in this appendix have been licensed for export across many borders. Two versions of implementation exist within the IBM Transaction Security System product range: full and commercial. The full version supports data integrity, key management, personal authentication operations, and general data confidentiality operations. The commercial version supports everything, including general data confidentiality operations, by using the Commercial Data Masking Facility (CDMF).

Usually, only customers in the USA and Canada, or subsidiaries of US-based companies that are outside the US, and financial institutions outside the USA and Canada can receive the products that have full capabilities. Generally, all other customers can receive the commercial version of the products. Consult with your marketing representative to understand how these restrictions can affect you.

A.9 PassTickets and Cryptography

This section describes the relation between PassTickets and various cryptographic operations.

A.9.1 PassTicket Algorithm

The PassTicket algorithm is a combination of the cryptographic function ENCRYPT and some arithmetic functions. All encryptions use the U.S. National Institute of Standards and Technology (NIST) Data Encryption Standard (DES) algorithm. Only the DES encrypt is involved. You cannot perform general encryption and decryption of data with this implementation. Refer to section A.2.1, “Data Encryption Algorithm” on page 81 for a short introduction of DES.

The key in this encryption is the application key (sometimes called the Secured Signon key) that is retrieved from, for example, the PTKTDATA profile. In normal cryptographic operations, the key is generated by a pseudo-random-number generator that is available in most cryptographic facilities. This technique can also be used to generate applications keys. The contents of the key must be 16 hexadecimal characters (64-bits). The only valid characters are 0 through 9 and A through F.

The PassTicket algorithm starts with two cryptographic operations. Input for the first encryption operation is the user ID that is used on the target application. The result of this operation is exclusive ORed with the profile name for the application in the PTKTDATA class. The result of this arithmetic operation is encrypted again with the same application key. This cryptographic process is similar to the process of the CCA Message Authentication Code algorithm.

The time-coder algorithm contains also one encrypt operation beside several arithmetic operations. The application key is again used as the cryptographic key.

If RACF is used to generate or validate PassTickets the standard DES encrypt routine that is currently available within RACF performs these cryptographic operations. NetSP also has a “ One way implementation of the DES algorithm” to perform these encrypt operations.

A.9.2 Encrypted Application Keys

Since the RACF PassTickets are, in most environments, *one-time-only* and *nonreusable* passwords, they do not need additional protection when they are sent across networks, or when they are (temporarily) stored in any data storage.

The secret application key, which is used to generate and validate PassTickets, must be protected against unauthorized viewing. Once this application key is exposed in any component in the network, it has to be replaced. Everyone that has to be able to generate PassTickets for a specific application must have access to the same application key, but they do not need to know the contents of that key.

This means that the application key has to be available in every system or server that is generating PassTickets for that application. This also implies that the application key must be transported to all the platforms where this key is going to be used.

Cryptographic facilities in the different network components can be used to protect the application keys from unauthorized viewing when they are sent across the network and also when they are stored in the network component.

The following cryptographic techniques can be used to provide additional security for the RACF Secured Signon facility:

- Data confidentiality in the network.

  By using VTAM Session Level Encryption (SLE), the secret application keys can be distributed across networks without the risk that unauthorized people can disclose the contents of the key.

  VTAM SLE implies automatic "Partner Validation." This ensures that the secret application key is transmitted to the right partner.

  Note: VTAM SLE does a single-encrypt under a single length key. Single encryption of keys is usually *not* recommended. Standard CCA key distribution *always* encrypts keys under double-length DES keys using a triple-encryption.

- Data confidentiality in another system or server.

  By using the encipher and decipher services, the secret application key can be stored under the master key of the cryptographic facility. Access control facilities in the cryptographic facilities can be used to protect the cryptographic services from unauthorized usage. By using these techniques, only the application that is generating the PassTicket can be authorized to access these cryptographic services.

With a cryptographic facility, RACF can store the secret application keys, which are needed to generate PassTickets, on the database in the form in which they are encrypted under the cryptographic product's master key. RACF uses the functions of the cryptographic facility to be sure the encrypted key does not exist in clear-text form within the system main storage for RACF processing, except when the keys are being defined. Therefore, if a system storage dump occurs, the keys are not exposed in the dump.

You should use the RACF RDEFINE command with PTKTDATA as the *class_name* operand to define RACF Secured Signon application profiles. For example, enter the following command:

```
RDEFINE PTKTDATA profile_name
    SSIGNON(KEYENCRYPTED(key_value))
    UACC(access_authority)
```

KEYENCRYPTED(*key_value***)** indicates that you want to encrypt the key value.

If you encrypt the application key on a system that is sharing the RACF database with other systems, but not all the systems are equipped with a cryptographic facility, you must ensure that the applications requiring the encrypted keys execute only on the systems with a cryptographic facility.

Installations that intend to use the KEYENCRYPTED option of the RACF Secured Signon must ensure that RACF can access the cryptographic callable services. To ensure this, the cryptographic callable services library should be concatenated to SYS1.LPALIB.

Appendix B. Network Security Program

The Network Security Program (NetSP) provides network security, identification, authentication, and key-distribution services to users and application programs using operating systems such as AIX, DOS, and OS/2, and transfer protocols such as TCP/IP, LU 6.2, and NetBIOS. The Network Security Program authenticates the identity of two communication principals and provides each with the ability to verify the identity of the other. The communication principals can be people (end users) or programs (*application servers*).

In a typical network, there are “client machines” representing users (both people and programmed users) that need access to the services or data provided by “application servers.” The application servers are application programs that maintain and control access to data or services, and must protect the data or service from unauthorized access. The client program in the client machine requests access to data controlled by the file-server program in the application server workstation.

The Network Security Program provides a trusted “third-party” security program whose function is to authenticate the credentials of both the user at the client machine and the program at the application server workstation, and to provide a means by which the user′s program and the application server program can identify and validate each other. This trusted third-party security program is called the Network Security Program *authentication server*.

The Network Security Program is convenient for users since they can logon once a day and be authenticated as a user of all the secured applications they are allowed to access. By logging on once to access multiple applications, the need to remember unique passwords for each application is eliminated.

The passwords used in this process are never sent across the network as readable text. For added security, all passwords are encoded before being transmitted over the network.

Network Security Program also allows you to reduce administrative complexity. By using a central *principal database* to contain clients and applications, you do not have to enter password information in every application the user is authorized to use. You can instead add the information once to the principal database stored on the authentication server.

The Network Security Program allows you to access RACF-protected applications on the host without sending unprotected passwords across the system. You can use the signon feature of the Network Security Program to access all the RACF-protected applications, and a unique PassTicket will be generated for each access of your host application or system.

Before you can use the signon facility of the Network Security Program, you must logon to the Network Security Program. Both the user and the RACF-protected application must be included in the Network Security Program principal database.

Network Security Program provides you with an audit facility that allows you to track authentication attempts. Failed authentication attempts are always registered in an audit-log file. In addition, you can have successful and failed

attempts logged. This ability to review the authentication attempts permits you to investigate any unusual activity that might indicate an attempted security breach.

The following sections describe the components of Network Security Program and the services they provide. These sections include only the components that are used to implement the Secured Single Signon facility described in this book.

B.1.1 Terminology

The following terms, used in this manual, may be unfamiliar to MVS and RACF oriented readers:

Client A computer or a process that accesses data, services, or resources of another computer or process in a network.

Server A computer or a process that contains programs, data, or provides services or other facilities other computers or processes on the network can access.

Initiator Of two communicating application programs, the one that initiates the authentication process.

Responder Of two communicating application programs, the one that is involved in the authentication process as a result of a request by the initiator

Principal An entry in the Network Security Program principal database. The authentication server uses the information in these principal entries to provide security credentials. Principals can be:

- Users
- Application servers
- Clients
- RACF-protected host applications of systems

Credential Keys created by the authentication server for all clients and application servers in the principal database. Security credentials are used to obtain authentication of session partners from the authentication server. There are two types of security credentials:

- Session keys
- Permanent keys

Session Key Temporary credentials obtained during logon. A session key expires after a specified number of hours. They are primarily used by clients, are stored in cache files at the client or application server workstation, and can be destroyed by logging off Network Security Program.

Permanent Key Obtained by using the “install key” function. They do not expire. For this reason, they are used most often for application servers. Permanent keys are stored in cache files. Even though permanent keys do not expire, they can be changed using a server “rekey” function.

Shared Key Key used to validate a session partner′s messages. When a client or an application server attempts to establish a secured session, it sends a request to the authentication server to authenticate its session partner. The authentication server

provides a shared key that each session partner can use to check incoming messages to ensure they are from the correct sender. The authentication server sends these shared keys to the client and application server in an encoded format that only that client and that application server can decode. The authentication server uses tickets to send these shared keys.

Ticket When an authentication server receives a request for authentication, it sends back a ticket. This ticket contains the shared key for the two principals establishing a session. The ticket is encoded and only the principal to which it is sent can decode it.

B.1.2 Authentication Server

The authentication server is the third party that provides security credentials to the principals entered in its principal database. When a client or an application server has to ensure that the session partner is a secured application or an authorized user, it can contact the authentication server. The authentication server uses the information in the principal database to provide the clients and application server with a means of checking the source of the messages they receive during the session.

The authentication server provides the audit facility in the Network Security Program.

B.1.3 Application Server

Workstations that have the application server component of the Network Security Program installed provide secured application services to the Network Security Program clients. The applications on these servers can be used only by clients that have been authenticated by the authentication server.

Application servers have permanent credentials and are entered as principals in the principal database.

B.1.4 Client Machine

Client machines are the entry points into the network for users and require services of the application servers.

The client portion of Network Security Program code allows clients to obtain credentials from the authentication server and, with input from the authentication server, validates the source of messages from the application servers protected by Network Security Program.

Client machines can have temporary credentials or permanent credentials and are stored as principals in the principal database.

B.1.5 Principal Database

The principal database is located on the authentication server. The identity and password of any client or application server that wishes to obtain security credentials must be entered as a principal in this database.

This database is also used to associate clients with the RACF-protected applications they are authorized to use. The principal database is protected by a

master password that must be entered to make any additions, deletions, or changes to the content of the database.

In order to access RACF-protected host applications, you must include the RACF user ID and the RACF application definitions from the RACF database into the principal database.

Note: By using the Secured Single Signon facility described in *Enhanced Exploitation of RACF 1.9.2 - Secured Single Signon Using NV/AS*, only the first session manager, NetView Access Services, is defined in the principal database. In the implementation of Secured Single Signon, an authorization check determines whether this request for a specific application in the host is a valid request for that user ID. Even if the requested application is running on a different host, but accessed by using NetView Access Services in this system, this request can be validated.

The principal database can be maintained by using standard Network Security Program commands.

B.1.6 Network Security Program Interface

There are three ways to access the functions of Network Security Program:

- Graphical User Interface (GUI)

  On OS/2 and AIX/6000, Network Security Program provides a graphical user interface for the client, application server, and authentication server. It is a panel-driven interface that allows you to request the functions of Network Security Program. Some of the functions it enables you to use are:

  – Logging onto Network Security Program and request the security credentials.
  – Signing onto RACF-protected host applications.
  – Working with the principal database.

- Command Line Interface

  Network Security Program provides a set of user commands that you enter on the command line of a client, an application server, or the authentication server. The command line interface provides the same functions as the GUI, plus additional functions, such as checking the time difference between system clocks.

  There are two sets of application program interface (API) calls provided by Network Security Program:

  – The Generic Security Services API (GSSAPI) provides the authentication functions.

  – The Network Security Management API (NSMAPI) provides the security credential management functions.

  Both GSSAPI and NSMAPI are supported only in C language programs.

  Note: The GSSAPI has been accepted for inclusion in Open Systems Foundation (OSF**) Distributed Computing Environment (DCE**) Release 1.1 and further extensions are planned for Release 1.2. X/Open is strongly endorsing GSSAPI with Internet Engineering Task Force (IETF).

B.1.7 Host Application Signon

Prior to starting any activity on his behalf, the user authenticates himself to the security system. By using a Network Security Program (NetSP) “login” command at the client machine, the user requests a session key from the authentication server. In this login sequence, the user is prompted for his NetSP password. The authentication server returns a session key in the cache at the client machine This key is valid until it expires or until the user log off from NetSP.

If the user is authorized by the authentication server to access any RACF protected application on the host, this NetSP login command creates two files containing RACF information. One file contains a list of RACF user IDs and system combinations that are associated with the NetSP user ID. Another file contains a list of all RACF application and system combinations stored on the authentication server. This file is stored in the same directory as the cache file, and is used by signon to any of the RACF-protected host applications.

After the user has identified himself to the authentication server and has obtained his session key, he can perform a signon to a RACF-protected host application by using a NetSP “signon” command at the client machine. This command can be given through the GUI by selecting a “signon host” icon; through the command line by issuing a NetSP signon command; or through an application program written using the NSMAPI.

There are two ways to perform the host signon. The user can use a so-called manual signon, or the user can replay an already recorded signon for this application. In this signon sequence, the user can supply information, such as the application name, the RACF user ID, that is used on this application, and the system name. This information can be defaulted from the two files that are created in the NetSP login command to the authentication server.

If the use of RACF Secured Signon is enabled for this host application, the signon can be tailored to request a PassTicket from the authentication server at the moment the signon sequence to the host application is requesting the password for that user ID on the specified application. The authentication server validates the request from this NetSP user ID for the combination of that specific host application on that specific system together with that specific RACF user ID. This check is made by using the information in the principal database.

The PassTicket is created by Network Security Program in the authentication server and sent back to the client code in the client machine. This client code stores the PassTicket in the signon data stream to the host application.

In this RACF Secured Signon implementation, only the authentication server has to know the secret application key that is needed for the PassTicket generation for the host application.

Note: The implementation of the NV/AS Secured Logon, as described in *Enhanced Exploitation of RACF 1.9.2 - Secured Single Signon Using NV/AS*, reduces the administration of the secret application keys to a minimum. Only the secret application key for the NV/AS application on the first host needs to be stored in the principal database.

B.1.8 Authentication Protocols

Authentication protocols are the methods of requesting authentication for session prompts and of receiving the tickets containing the shared key. Which protocol you choose depends on whether you want the client or the application server to be connected to the authentication server.

Basically, there are only two types of security protocols:

- One-way Authentication

  In one-way authentication, the identity of the client is always authenticated to the application server, but the identity of the application server is never authenticated to the client.

- Mutual Authentication

  In mutual authentication, the identity of the client is authenticated to the application server and the identity of the application server is authenticated to the client. In mutual authentication, the authentication server sends two pieces of security information. One piece goes to the client, can only be understood by the client, and authenticates the identity of the application server. The other piece goes to the application server, can only be understood by the application server, and authenticates the identity of the client.

When the user at the client machine wants to signon to a host RACF-protected application, a one-way authentication protocol is used.

B.1.9 Network Security Program Administration

Network Security Program administrative functions can be divided into the following groups:

- Functions for all components of the Network Security Program
- Functions for the authentication server, such as:
  - Creating, backing up, or restoring the principal database
  - Verifying the contents of the principal database
  - Adding or deleting principals to the principal database
  - Changing principal passwords
  - Auditing authentication activity
- Functions for the application server
- Functions for the client machine

The first step in setting up the Network Security Program is to create the principal database to hold all the entries for users and application servers. The principal database can be created on the authentication server using a NetSP command.

By creating the database, you will be prompted for a master password to be associated with this principal database. This master password is stored in the principal database in encrypted format. If you do not stash the password, you will be prompted for the master password each time you attempt to edit the database.

If there are many RACF user ID′s to add to the NetSP principal database, you may want to run the RACF database unload utility IRRDBU00 to create a flat file containing the information from the RACF database. Transfer this file to the NetSP environment by using a download function, such as FTP.

The flat file can be converted using a NetSP command into a format that can be edited and loaded into the principal database. If you have downloaded multiple files from multiple host systems, you can run the same command and append the output together.

The Network Security Program principal name included in the file is based on the RACF user ID. The NetSP password is initially the first eight characters of the principal name.

You can edit the file that is produced by the NetSP command to change the passwords, or to associate multiple RACF user ID′s with one NetSP principal name by entering the same principal name for several different RACF user ID′s.

Once you have edited the file, you can load the RACF information into the principal database by using another NetSP command.

You need to add the RACF-protected applications to the principal database individually. For each RACF-protected application, you need to provide the name and host system to the principal database as well as application-specific data, such as the secret application key. This key must match the corresponding RACF host entry on the RDEFINE statement for the profile that describes the application on the MVS host.

For a complete overview of the administration functions available, refer to *Network Security Program Developer′s Guide*.

Appendix C. Applications of the Personal Security Card

Chapter 4, “IBM Personal Security Card” on page 19 has given a brief overview of the IBM Personal Security card. The card has a highly flexible architecture, making it easy for customers to adapt it to meet the needs of many different applications. The security features allow cryptographic functions, sophisticated user authorization checking, and protection of data blocks stored in the card′s EEPROM memory. The card is an integrated part of the IBM Transaction Security System, with a completely compatible cryptographic architecture.

The Personal Security card is not limited to traditional cryptographic applications, nor to traditional smart card applications. Its unique features result in applications mixing the cryptographic and smart card environments, and extending into new applications even the designers did not envision.

Sample applications of the IBM Personal Security card are suggested by the data block usages as described in the following section.

C.1 How Data Blocks on the Personal Security Card Can Be Used

Data blocks on the Personal Security card are very flexible and lend themselves to a wide variety of uses. The list of examples below will help you understand their power.

- Data blocks may be used for information about the cardholder. This could include such things as the user′s name, address, phone number, employer, social security number, and bank account number.
- Data blocks can hold authorization information used by the application program that runs in the workstation or on the host. The program can read a data block and use its contents to decide what application program functions the user is permitted to execute.
- The data blocks can be used to store a variety of medical data that could include medical history, prescriptions to be filled, or images of X-Rays.
- The smart card can be used as an ″electronic wallet″. In this application, cash is withdrawn from the user′s account at an ATM and the amount is stored in a data block on the smart card. The money can be spent at any suitably equipped store, vending machine, pay telephone, and so forth, where the amount spent is subtracted from the balance in the card.
- The smart card can be used to transport cryptographic keys or key parts. Data blocks can be used to transport keys to remote locations, where they are installed in a 4753, 4754, or 4755. Keys can also be transported using the card′s internal key tables, of course.
- The card can be used to hold digitized images from a scanner or video source. A compressed, digitized image of the cardholder can be stored in a data block, as can a scanned image of the cardholder′s signature. Either of these can be read and displayed to help verify the user′s identity as the owner of the card.
- Data blocks can be used for transaction logs, to record the use of the smart card. The application program can log whatever information is important in tracking the cardholder′s transactions.

- Data blocks can be defined for any application-specific data that relates to the card user. All applications use different data, and it is frequently advantageous for some of that data to be on the smart card. This makes the data portable, so it is always available at any workstation. It also associates the data automatically with the owner of the card.
- In addition to these customer applications, data blocks are also used by some TSS system functions. When the signature verification feature is used, the signature reference data is stored in data blocks on the smart card. The user′s reference data is read out by the 4755 Cryptographic Adapter, where it is compared with the signature data from the pen.

As you can see, the applications of data blocks are almost limitless. This list was included to stimulate your imagination about possible uses in other applications.

C.2 Personal Security Card Applications

This section describes more example applications, and provides more detail on some already mentioned. The applications require varying degrees of security, but most use several of the smart card′s security features.

C.2.1 Electronic Funds Transfer (EFT)

EFT is intended to be the main application of the Transaction Security System. The Personal Security card does several functions in this environment.

- The profiles on the smart card control user authorization, both in the card and in the 4754 and 4755. The card can be used to authorize generation of MACs, verification of MACs, encryption of data, release of payment transfers, and so forth.

  Note: MAC is an acronym for **Message Authentication Code**. It is a cryptographically calculated code sent with a message, which is used to verify the message has not been altered.
- The smart card can carry cryptographic keys used in the EFT application. The cryptographic functions might be performed in the smart card itself, or in the 4754 or 4755. Storing keys in the smart card makes it easy for each user to have individually assigned keys.
- The smart card can actually perform the cryptographic processing needed in the EFT application. Typically, this will involve generating or verifying MACs for payment transfer messages.

C.2.2 Medical Applications

Several possible applications use the smart card to carry medical information. It can carry medical history information, recorded each time you visit a doctor. Every doctor will be able to see a complete medical history by examining your card. The card can be used to carry prescriptions. The doctor can store the prescriptions in the card, and the pharmacist can store records indicating when each was filled. The card can carry information about important medical conditions, like those shown on medical bracelets today. Finally, the smart card could be used to carry information scanned from images such as an X-Ray. This application is limited, however, due to the amount of EEPROM memory in the card.

C.2.3 Portable Wallet

As mentioned earlier, the portable wallet application is a popular concept. A data block on the card is loaded with some amount of cash at an ATM, and the cash is decremented as the user makes purchases. Cards might be accepted at retail stores, gas stations, vending machines, pay phones, and so forth. At any time, the user can stop at an ATM and add more cash to the balance in his card.

C.2.4 Host Computer Logon and Applications

The smart card can carry data used when the user accesses a host computer from a workstation. The card could hold the node ID of the owner′s host system, and his user ID and possibly his password. The logon process could be automated, only requiring the user to insert his card in the reader and verify his PIN or signature. The card could also hold the name of an application to run once the user logs on, or a list of valid applications the user is permitted to access on the host.

C.2.5 Electronic Passport

All the information in a passport could be stored in the smart card, including a digitized color photograph, a digitized signature, and all information about the owner and his trips. This would be far more resistant to fraud than a conventional passport.

C.2.6 Cryptographic Key Distribution

The smart card can be used as a secure and convenient way to transport keys to a cryptographic device. The TSS utility software supports this capability. Data keys, key encrypting keys, and master keys can all be transported on the smart card. There are several different possibilities.

- Master keys and key encrypting keys are sometimes loaded into a cryptographic device using clear text key parts, where each part is supplied by a different person. The key parts can be stored on smart cards which are sent to each of those people.
- If the smart card and the target device share a common key encrypting key, the keys to be distributed can be loaded into the smart card′s internal key tables. The card can export the keys to the target device, encrypted under the common key encrypting key.
- The keys can be encrypted outside the smart card, using a key not known to the card. The encrypted keys can then be stored in data blocks on the card. At the destination, the keys are read from the data blocks and decrypted.

C.2.7 Electronic Benefits Distribution

The smart card can be used to distribute benefits, like food stamps, in place of the paper coupons used today. The security features of the card can eliminate most of the fraud that constantly plagues such programs. Each period, the user′s allotted benefits would be added to the card. The balance would be decremented as it is used. In many ways, this is similar to the electronic wallet application.

In a sophisticated implementation of this concept, the electronic cash register could be integrated into the system. The register could log all purchases to the smart card, or it could refuse to allow the purchase of certain items, such as alcoholic beverages.

C.2.8 College ID Card

A smart card could serve several purposes in an environment such as a college campus. The card could act as a student ID and a cafeteria meal card, and control access to dormitories and other buildings. Similar applications exist in industry, where magnetic stripe ID badges are used today.

C.2.9 Multiple Applications

Often, a single smart card can be used for more than one application. Each application can have its own data blocks and cryptographic keys. The use of secret block tokens or secured data blocks can prevent one application from accessing another application's data blocks.

Index

Special Characters

Numerics

A

B

C

D

E

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

ITSO Technical Bulletin Evaluation RED000

Secured Single Signon in a Client/Server Environment

Publication No. GG24-4282-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction	____		
Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

_____

_____

What other topics would you like to see in this Bulletin?

_____

_____

What other Technical Bulletins would you like to see published?

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____ _____
Name Address

_____ _____
Company or Organization

_____ _____
Phone No.

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Mail Station P099
522 SOUTH ROAD
POUGHKEEPSIE NY
USA 12601-5400

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A.